

# مدخل إلى الخوارزميات

الجزء الأول

INTRODUCTION TO  
ALGORITHMS



# مَدْخُلٌ إِلَى الْخَوَارِزْمِيَّاتِ

**Introduction to Algorithms**

*Third Edition*

الجزء الأول

# مَدْخُلٌ إِلَى الْخَوَارِزْمِيَّاتِ

## الإصدار الثالث

تأليف

Thomas H. Cormen

Charles E. Leiserson

Ronald L. Rivest

Clifford Stein

ترجمة

د. علي أبو عمشة د. محمد سعيد الدسوقي

د. أميمة الدكاك د. ندى غنيم

د. كمال قمر د. غيداء ريداوي

د. رضوان قسطنطين



حقوق الطبع محفوظة للجمعية العلمية السورية للمعلوماتية

2012

عنوان الكتاب الأصلي

# **Introduction to Algorithms**

## **THIRD EDITION**

**Thomas H. Cormen**  
**Charles E. Leiserson**  
**Ronald L. Rivest**  
**Clifford Stein**

**The MIT Press**

**Cambridge, Massachusetts   London, England**



# نظرة سريعة إلى محتوى الجزء الأول

## مقدمة

### الباب الأول أساسيات

#### تمهيد

- 1 دور الخوارزميات في الحوسبة
- 2 لبدأ
- 3 نمؤ الدوال
- 4 فرق-تسد
- 5 التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة

### الباب الثاني الفرز وإحصائيات الترتيب

#### تمهيد

- 6 الفرز بالكومة
- 7 الفرز السريع
- 8 الفرز في زمن خطي
- 9 الأوساط وإحصائيات الترتيب

### الباب الثالث بنى المعطيات

#### تمهيد

- 10 بنى المعطيات الأولية
- 11 جداول التليد
- 12 أشجار البحث الثنائية

13 الأشجار الحمراء-السوداء

14 إغناء بني المعطيات

---

#### الباب الرابع تقنيات متقدمة في التصميم والتحليل

---

تمهيد

15 الهرمجة البنيائية

16 الخوارزميات الشرهة

17 تحليل الكلفة المختدة

---

#### الباب الخامس بني المعطيات المتقدمة

---

تمهيد

18 الأشجار المعممة

19 كومات فيوناتشي

20 أشجار Van Emde Boas

21 بني المعطيات للمجموعات المنفصلة

## نظرة سريعة إلى محتوى الجزء الثاني

### الباب السادس خوارزميات البيانات

#### تمهيد

- 22 خوارزميات البيانات الأساسية
- 23 أشجار المسح الصغرى
- 24 أقصر المسارات من منبع وحيد
- 25 أقصر المسارات بين جميع أزواج العقد
- 26 التدفق الأعظمي

### الباب السابع مواضيع مختارة

#### تمهيد

- 27 الخوارزميات المتعددة النيات
- 28 العمليات على المصفوفات
- 29 البرمجة الخطية
- 30 كثيرات الحدود وتحويل فورييه السريع
- 31 خوارزميات نظرية الأعداد
- 32 مطابقة متاليات المحارف
- 33 الهندسة المحوسبة
- 34 تعقيد المسائل
- 35 خوارزميات التقريب

---

## الباب الثامن الملاحق: معارف رياضية أساسية

---

تمهيد

الملحق أ المجموع

الملحق ب المجموعات ومفاهيم أخرى

الملحق ت العد والاحتمالات

الملحق ث المصفوفات

---

مسرد المصطلحات عربي - إنكليزي

مسرد المصطلحات إنكليزي - عربي

المراجع

الفهرس

# محتوى الجزء الأول

## مقدمة xxv

### الباب الأول أساسيات

#### تمهيد 3

#### 1 دور الخوارزميات في الحوسبة 5

1.1 الخوارزميات 5

2.1 الخوارزميات بصفتها تقانة 11

#### 2 لنبدا 16

1.2 الفرز بالإدراج 16

2.2 تحليل الخوارزميات 23

3.2 تصميم الخوارزميات 30

#### 3 نمؤ الدوائ 44

1.3 التدوين المقارب 44

2.3 تدوينات قياسية ودوال شائعة 55

#### 4 فرق-تسد 67

1.4 مسألة الصفيغة الجزئية العظمى 69

2.4 خوارزمية شتراسن لجداء للصفوفات 77

3.4 طريقة التعويض لحل العلاقات القؤدية 85

4.4 طريقة شجرة القؤدية لحل العلاقات القؤدية 90

5.4 الطريقة الرئيسة لحل العلاقات القؤدية 95

6.4 \* برهان للمبرنة الرئيسة 99

#### 5 التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة 116

1.5 مسألة التوظيف 116

2.5 المتحولات العشوائية للمؤشرة 119



3.5	الخوارزميات ذات العشوائية المضافة	124
4.5	التحليل الاحتمالي واستخدامات إضافية للمؤشرات العشوائية	131

\*

## الباب الثاني الفرز وإحصائيات الترتيب

### تمهيد 149

#### 6 الفرز بالكومة 153

1.6	الكومات	153
2.6	الحفاظ على خاصية الكومة	156
3.6	بناء كومة	158
4.6	خوارزمية الفرز بالكومة	161
5.6	الأوتال ذات الأولوية	163

#### 7 الفرز السريع 171

1.7	وصف الفرز السريع	171
2.7	أداء الفرز السريع	175
3.7	نسخة للفرز السريع ذو عشوائية مضافة	180
4.7	تحليل الفرز السريع	181

#### ■ الفرز في زمن خطي 192

1.8	الحدود الدنيا للفرز	192
2.8	الفرز بالعد	195
3.8	الفرز حسب الأسس	198
4.8	الفرز بالدلاء	201

#### 9 الأوساط وإحصائيات الترتيب 214

1.9	الأصغر والأكبر	215
2.9	الاختبار بزمن خطي متوقع	216
3.9	الاختبار بزمن خطي في أسوأ الحالات	220

## الباب الثالث بنى المعطيات

### تمهيد 229

#### 10 بنى المعطيات الأولية 233

- 1.10 للكُلمات والأرتال 233
- 2.10 اللوائح المترابطة 237
- 3.10 تنجيز المؤشرات والأغراض 242
- 4.10 تمثيل الأشعار ذوات الجذور 247

#### 11 جداول التلييد 254

- 1.11 جداول العنوان المباشر 255
- 2.11 جداول التلييد 257
- 3.11 دوال التلييد 263
- 4.11 العنونة للمفتوحة 271
- 5.11 \* التلييد الكامل 279

#### 12 أشجار البحث الثنائية 287

- 1.12 ما هي شجرة البحث الثنائية؟ 287
- 2.12 استعلام شجرة بحث ثنائية 291
- 3.12 الإدراج والحذف 295
- 4.12 \* أشجار بحث ثنائية مبنية عشوائيًا 301

#### 13 الأشجار الحمراء-السوداء 310

- 1.13 خصائص الأشجار الحمراء-السوداء 310
- 2.13 الدورانات 314
- 3.13 الإدراج 317
- 4.13 الحذف 324

#### 14 إغناء بنى المعطيات 340

- 1.14 إحصائيات الترتيب الديناميكية 340
- 2.14 كيف نغني بنية معطيات 346
- 3.14 أشجار الحالات 350

---

**الباب الرابع تقنيات متقدمة في التصميم والتحليل**


---

**تمهيد 359****15 البرمجة الديناميكية 361**

- 1.15 تقطيع الفضاء 362
- 2.15 جداء سلسلة من المصفوفات 372
- 3.15 عناصر البرمجة الديناميكية 380
- 4.15 أطول متتالية جزئية مشتركة 392
- 5.15 شجرات البحث الثنائية للثلي 399

**16 الخوارزميات الشرهة 417**

- 1.16 مسألة اختيار النشاطات 418
- 2.16 عناصر الاستراتيجية الشرهة 425
- 3.16 أرزمة هوفمان 431
- \* 4.16 الكيانات للمصفوفية والطرائق الشرهة 439
- \* 5.16 مسألة جدولة المهام 446

**17 تحليل الكلفة المعقدة 454**

- 1.17 التحليل التجميع 455
- 2.17 طريقة المحاسبة 459
- 3.17 طريقة الكمون 462
- 4.17 الجداول الديناميكية 466

---

**الباب الخامس بنى المعطيات المتقدمة**


---

**تمهيد 485****18 الأشجار المعشمة 488**

- 1.18 تعريف الأشجار للمعشمة 492
- 2.18 العمليات الأساسية على الأشجار للمعشمة 495
- 3.18 حذف مفتاح من شجرة معشمة 503

<b>19</b>	<b>كومات فيوناتشي 510</b>
1.19	بنية كومات فيوناتشي 512
2.19	عمليات الكومات القابلة للدمج 515
3.19	إنقاص قيمة مفتاح وحذف عقدة 523
4.19	وضع حد للدرجة العظمى 527
<b>20</b>	<b>أشجار Van Emde Boas 536</b>
1.20	منهجات مبدئية 537
2.20	بنية عودية 541
3.20	شجرة Emde Boas 551
<b>21</b>	<b>بنى المعطيات للمجموعات المنفصلة 566</b>
1.21	عمليات المجموعات المنفصلة 566
2.21	تمثيل المجموعات المنفصلة بـالاتحة مترابطة 569
3.21	غابات المجموعات المنفصلة 573
4.21	تحليل الاجتماع بحسب المرتبة وضغط المسار 577
	★



# كلمة الجمعية العلمية السورية للمعلوماتية

عزيزي القارئ

نضع الجمعية العلمية السورية للمعلوماتية اليوم بين يديك، وبعد طول انتظار هذا الكتاب "مدخل إلى الخوارزميات"، الذي عَمِلَ على نقله إلى العربية مجموعة متخصصة من الباحثين والمدرّسين في علوم الحاسوب عموماً، والخوارزميات خصوصاً. ويأتي هذا الكتاب في إطار جهود الجمعية المستمرة لإغناء المكتبة العربية بكتب تخصصية في المعلوماتية بلغة عربية سليمة ومعاصرة تساعد القارئ العربي على الحصول على أكثر العلوم معاصرة بلغته الأم.

وقد قامت لجنة التأليف والترجمة والنشر في الجمعية منذ العام 2000، بترجمة مجموعة من كتب المعلوماتية المتميزة، التي يعدُّ بعضها مرجعاً أساسياً لطلاب الجامعات. نذكر منها: "أسس لغات البرمجة" المنشور عام 2000، و"هندسة البرمجيات" منمّج للممارس" (في جزأين) عام 2001، و"الدكاء الصناعي" عام 2004، و"مفاهيم نظم التشغيل" (في جزأين) عام 2005، و"التعمية التطبيقية" و"اتصالات المعطيات والحواسيب" عام 2006، إضافة إلى منشورات أخرى لا تقلُّ عنها أهمية مثل: معجم مصطلحات المعلوماتية الذي نُشر في العام 2000، والذي يقع الآن في صُلْب مشروع عربي بالتعاون مع الاتحاد العلمي للاتصالات لإصدار نسخة موسّعة وتحديثه منه.

يمالِج هذا الكتاب أحد أهم الأسس في علوم الحاسوب، ألا وهو بنى المعطيات والخوارزميات. ولا يقتصر هذا الكتاب - كغيره من كتب الخوارزميات العديدة - على كونه "كتاب وصفات" يبيّن أهمّ ما استقرّت عليه الدراسات والبحوث في هذا المجال، بل يأخذ بيد القارئ خطوة خطوة؛ فيشرح كلّ مسألة بالتفصيل، ثمّ يعرض مجموعة من الحلول ويقارن بينها مستعيناً بأدوات الرياضيات المتقطّعة التي لا غنى عنها للوصول إلى فهم عميق لهذه الحلول، ويرمي هذا الأسلوب إلى تطوير قدرة الدارس تدريجياً على المقارنة والنقد واختيار وتصميم الحلول الفضلى للمسائل التطبيقية التي قد تعترضه.

إن النسخة الأصلية من هذا الكتاب "Introduction to Algorithms" هي كتاب مرجعيّ من منشورات دار نشر معهد ماساتشوستس للتقانة MIT Press. وهو مرجعٌ تدريسيّ معتدّ في معظم جامعات العالم، ويعدُّ من أكثر الكتب المرجعيةّ مبيّهاً؛ فقد بلغ عدد النسخ للمبيعة منه حتى آب 2011 نصف مليون نسخة<sup>1</sup>، وذلك منذ إصداره الأول في العام 1990. وهو إلى ذلك كتاب شاملٌ يقدّم العون للطلاب من بداية دراسته الجامعية وحتى دراساته العليا. ومما بدّل على كبير أهمية هذا الكتاب وعلوّ شأنه في بايّه أنه وُزِعَ في مراجع ما يزيد على خمسة آلاف بحثٍ باعتباره أحد المراجع الأساسية في الخوارزميات، إضافةً إلى أنه مرجع لا يُستغنى عنه في جميع مجالات علوم الحاسوب<sup>2</sup>.

<sup>1</sup> <http://web.mit.edu/news/office/2011/introduction-to-algorithms-500k-0810.html>

<sup>2</sup> <http://citeseerx.ist.psu.edu>

يتجاوز عند صفحات الكتاب الأصلي 1200 صفحة، ولهذا السبب رأى فريق التعريب إصدار النسخة العربية في جزأين (مقتارين في عدد الصفحات) لتسهيل استعماله. يتضمن الجزء الأول أساسيات تحليل الخوارزميات، وخوارزميات الفرز، وبنى للمعطيات الأساسية، وبنى معطيات متقدمة، وبعض الطرق المتقدمة في حل المسائل. ويضم الجزء الثاني - الذي سيصدر لاحقاً - خوارزميات نظرية البيان، إضافة إلى تسعة فصول تعالج مواضيع مختارة ذات أهمية كبيرة. ويحتتم الكتاب مجموعة من الملاحق في مواضيع رياضية ذات صلة وثيقة بالخوارزميات، ثم قائمة مراجع غنية تزيد على 350 مرجعاً، ومصادر تبين المصطلحات باللغتين العربية والإنكليزية.

مولفوا الكتاب بدءاً من إصداره الثاني أربعة: اثنان منهم مدرّسان في معهد التقنية في ماساتشوستس، وهما Ron Rivest و Charles E. Leiserson، والأعوان من طلاب الدراسات العليا اللامعين في المعهد في الشمانينيات، وهما Clifford Stein و Thomas H. Cormen. ويجتهد الآن من العلماء المرموقين، وهم إسهامات في العديد من مجالات علوم الحاسوب. فعلى سبيل المثال، قدّم Rivest العديد من الإنجازات في مجال علم التعمية، وهو أحد مطوّري خوارزمية RSA الشهيرة. وزميله Leiserson إسهامات كثيرة في الحوسبة التفرعية والحوسبة الموزعة، وهو أحد رؤّاد تطوير نظرية VLSI، ومعظم شبكة الوصل fat tree المستعملة في الحواسيب الفائقة. أما Cormen، فهو مدرّس وكاتب بارع، تعدّد إنجازاته في مجال التعليم الجامعي، وهو رئيس قسم برنامج الكتابة في جامعة دارغوث. وأما Stein، فقد أسهم في تأليف كتاب "مدخل إلى الخوارزميات" بدءاً من إصداره الثاني، وهو مختصّ في بحوث العمليات، ويرأس حالياً قسم الهندسة الصناعية وبحوث العمليات في جامعة كولومبيا، وقد أغنى الكتاب بالعديد من الأمثلة التطبيقية.

أما فريق التعريب فقد ضمّ نخبة من خيرة الباحثين والمدرّسين ذوي الخبرة الطويلة في المعهد العالي للعلوم التطبيقية والتكنولوجيا وفي جامعة دمشق، وهم: د. علي أبو عمنه، و د. محمد سعيد دسوقي، و د. أميمة الذكّاك، و د. ندى غنيم، و د. كمال نصر، و د. غيداء ريداي، و د. رضوان قسطنطين. وقد قاموا معاً بعمل تعاوني ضخم - ضمن فريق واحد متكامل - تحلّل في تعريب فصول الكتاب، كلّ منهم فيما هو أقرب إلى خبرته واختصاصه، ثم تداولوا مراجعة هذه الفصول فيما بينهم. وبعد ذلك، قام الأستاذ مروان البواب مشكوراً بالمراجعة اللغوية وإدخال التعديلات الناتجة عنها. وكانت د. علي أبو عمنه هي المسؤولة عن تنسيق العمل بين أعضاء الفريق، وعن تعريب المصطلحات العلمية وتوحيدها، وأحرزت لهذا الغرض سلسلة من المناقشات مع زملائها. وأخيراً قام د. رضوان قسطنطين بعمل مهم يُدعى بحق نتيجاً لهذا الكتاب؛ فنسق نصوصه، وضبط معادلاته، ورَتب مقاطعه البرمجية، وأخرجها بحلّة أنيقة ثماني النسخة الأصلية له.

ويطيب لنا أن نشكر أفراد أسرة العمل كافة على الجهد الكبير الذي بذلوه في التعريب والمراجعة العلمية واللغوية. ونخص بالشكر د. علي أبو عمنه - التي كانت المحرّك الأول للانطلاق بهذا العمل - على جهودها في التنسيق، وفي العمل على توحيد للمصطلحات في هذا الكتاب الضخم. ونتوجه شكراً خاصاً أيضاً إلى د. رضوان قسطنطين على عنايته الكبيرة في إخراج النص وإدخال الأشكال والمعادلات الكثيرة لتظهر بأسلوب موحّد في كامل النص، وعلى جهوده للمحافظة على توحيد للمصطلحات. ونشر هنا إلى أن د. قسطنطين - إضافة إلى أنه تدارك الملاحظات التي عثر عليها في

أثناء إخراجهِ للكتاب - قام بتصحيح الأخطاء المنشورة على موقع الكتاب<sup>3</sup> حتى تاريخ الانتهاء من إعداد نسخته للمعربة. والشكر موصولٌ كذلك إلى الأستاذ مروان اليؤاب على جهوده في مراجعة هذا الكتاب، التي شملت الجوانب اللغوية والعلمية أيضاً، بعناية ودقة كبيرتين. وقد كانت له ملاحظاتٌ دقيقةٌ فيما يخصُّ تعريبَ العديد من المصطلحات. وفي الختام نتوجّه بالشكر الجزيل إلى الأستاذ الدكتور رakan رزوق، رئيس مجلس إدارة الجمعية العلمية السورية للمعلوماتية على دعمه للجنة الترجمة والتأليف والنشر، وتشجيعه على أن يرى هذا الكتاب النور. وأخيراً نأمل أن نكون بكتابنا هذا قد وفّقنا في وضع ما يساعد على فهم مواضيع اتصالات المعطيات والحواسيب بين يدي القارئ العربي. ونرجو أن يصدر الجزء الثاني منه في القريب العاجل.

والله ولي التوفيق.

الأستاذ الدكتور وائل معلا  
رئيس لجنة الترجمة والتأليف والنشر  
في الجمعية العلمية السورية للمعلوماتية





# كلمة فريق التعريب

زميلنا المدرّس

عزيزنا الطالب

نقدّم لك الجزء الأول من النسخة العربيّة للكتاب المرجعيّ الشهير "Introduction to Algorithms"، وبإذن الله سيصدر الجزء الثاني منه بعد مدّة غير طويلة.

إن هذا العمل ثمرّة جهود فريقنا التي امتدّت عدّة سنوات. ولن يخفى عليك، أيّها القارئ العزيز، عندما تبدأ بقراءة صفحات هذا الكتاب مدى الجهود المبذولة من مؤلّفي الكتاب لجعله شاملاً وواضحاً ودقيقاً، ومن فريقنا الذي بذل قصارى جهده ليقدم لك عملاً علمياً رائعاً بلغة عربيّة سليمة وبسيطة، ولينقل الأفكار وحتى أسلوب المؤلفين بأقصى قدر من الأمانة، راجين بذلك أن نتيح لقارئنا العربي الاطلاع على منشورات قيّمة بلغته الأم، عسى أن تصبح أمنا العربيّة من جديد متّاحة للعلوم ومطوّرة لها.

تعود جذور هذا الكتاب إلى منتصف السبعينيات، وكان قوائمّه وتحتوي محاضرات في الخوارزميات أُلقيت في معهد MIT. وفي منتصف الثمانينيات شجعت براءة Cormen في الكتابة العلميّة أساتذته آنذاك على حوضي مغامرة تأليف كتاب صدر في العام 1990، وزاد عدد صفحاته على 1000 صفحة، وصار مع مرور الأيام المرجع الأساسي للخوارزميات في العديد من جامعات العالم. وقد بدأت صلة فريقنا بالكتاب باعتداده مرجحاً أساسياً في تدريس مقرّر الخوارزميات في المعهد العالي للعلوم التطبيقية والتكنولوجيا وفي جامعة دمشق، وكنا نستمتع بعميقه ودقّته واستخدامه للرياضيات بالقدر المناسب لتعميق مفاهيم الخوارزميات. ونشأت بذلك فكرة نقله إلى العربيّة لنشارك به طلابنا وكلّ من يحوّله حاجز اللغة عن الاستفادة من هذا الكتاب. وقد قُبِلَت الجمعية العلميّة السوريّة للمعلوماتية مشكورة اعتماد هذا المشروع ضمن جهودها لإغناء المكتبة العربيّة التقنيّة بالكتب القيّمة.

وقد تكون فريقنا لإنجاز مهمّة تعريب هذا الكتاب في العام 2007. وكان أول ما بدأنا به هو تعريب مشرّط المصطلحات لضمان توحيدها في جميع فصول الكتاب، على الرغم من تعدّد مترجمي هذه الفصول. وقد اعتمدنا في ترجمة هذه المصطلحات في المقام الأول على ما ورّد في معجم مصطلحات للمعلوماتية الصادر عن الجمعية عام 2000، واحتفظنا - في العديد من الاجتماعات وبالتعاون مع المدقّق اللغوي - في ترجمة المصطلحات التي لم تردّ في المعجم. وعلى الرغم من كلّ الجهود المبذولة في هذا الصدد، فقد يكون هناك أكثر من مقابل بالعربيّة لبعض المصطلحات الإنكليزية، إلا أننا نعتقد أنه لن يكون لهذا كبير أثر على فهم القارئ لمضمون الكتاب، وذلك بفضل ما تتمتع به فصول الكتاب من استقلاليّة فيما بينها.

ونجدر الإشارة إلى أن عملنا بدأ بتعريب الإصدار الثاني لهذا الكتاب، وبعد أن قاربنا على الانتهاء من الجزء الأول منه، غلّطنا بصور الإصدار الثالث، وكان ذلك في آب 2009. وقد احتوى هذا الإصدار الجديّد تحديثات كثيرة حقّقته

أكثر قُرْبًا من القارئ، إضافةً إلى تدارك بعض النقاط التي كانت معالجةً بطريقةً مختلفةً عما كان دارجًا في كتب الخوارزميات الأخرى، وأُغْنِيَتْ تلاحُظٌ بحيث لا يحتاج القارئ إلى العودة إلى كتب الرياضيات لاستدراك المفاهيم اللازمة لفهم النص. فلم يكن بوسع فريقنا أن نحرّم النسخة العربية من هذه التحديثات، لاسيما أن مؤلفي هذا الكتاب يعملون سنواتٍ عدّة قبل إطلاق إصدارٍ جديدٍ له، فاختدنا قرارًا صعبًا بمراجعة جميع الفصول المعرّبة وتحليلها مطابقةً للإصدار الثالث.

يعالج الكتاب الخوارزميات التي تقع في الواقع في صُلْب علم الحاسوب وتطبيقاته، فيبدأ بالتعريف بالخوارزميات ويجانب تحليلها المتعمّد، ثم يقدّم في كلّ فصل: بنية معطيات مع الخوارزميات المتعلقة بها، أو تقنية تصميم، أو مجالًا تطبيقيًا، أو موضوعًا ذا صلة بمنوان الفصل. وقد راعى المؤلفون أن تكون الفصول مستقلةً فيما بينها قدر الإمكان، لتكون قراءة أيّ فصلٍ سهلةً وواضحةً دون الحاجة الكبيرة إلى الاطلاع على الفصول التي سبقته، ما عدا، ربما، فصول الباب الأول التي تُعنى بالأساسيات في تحليل الخوارزميات، أو للملاحق لتلافي الثغرات في الأساس الرياضي للقارئ.

يقع الكتاب في خمسةٍ وثلاثين فصلًا موزعةً على سبعة أبواب. يضمُّ الجزء الأول من النسخة المعرّبة خمسة أبواب:

- يزوّد الباب الأول القارئ بالسياق والأدوات اللازمة للمضي قُدُمًا في الكتاب. فهو يعالج مبادئ تصميم الخوارزميات وتحليلها. ويضمُّ مقدمةً سهلةً في توصيف الخوارزميات والأدوات اللازمة لذلك، ثم يُعرّف بعض استراتيجيات التصميم المهمة المستعملة لاحقًا في الكتاب، وخاصةً استراتيجية "فرّق-تحدّد".
- يعالج الباب الثاني مسألة فرز الأعداد التي تقع في صُلْب معظم التطبيقات المحوسبة، ويقدم خوارزميات عديدةً لحلّ هذه المسألة، ويدرس أدائها. ويعالج الفصل الأخير من هذا الباب ما يُعرف بإحصائيات الترتيب.
- يعرض الباب الثالث بعض بني للمعطيات والتقنيات الأساسية لتمثيل مجموعات للمعطيات الديناميكية المنتهية، وكيفية التعامل معها حاسوبيًا من استفسارٍ وتعديلٍ وتنظيم. تشمل بني للمعطيات المدروسة في هذا الباب البنى البسيطة مثل: للكنس، والرتّل، والقائمة للترابطة، والشجرة ذات الجذر. ثم ينتقل إلى جداول التّشبيد والأشجار الثنائية والأنواع المُنتهية منها.
- يدرس الباب الرابع ثلاث تقنيات هي: الترجمة الديناميكية، والخوارزميات الشرهة، والتحليل المخمّد. وتضاف هذه التقنيات إلى تلك التي يقدّمها الباب الأول، وهي تُستعمل بكثرة في تصميم الخوارزميات الفعالة وتحليلها.
- الباب الخامس مخصّصٌ لدراسة بعض بني للمعطيات المتقدّمة التي تدعم العمليات على المجموعات الديناميكية بفعالية أكبر من البنى المدروسة في الباب الثالث، لكنها أكثر تعقيدًا، إذ تُستعمل بعض البنى المدروسة بكثافةٍ تقنيات التحليل للمخمّد التي يعالجها الباب الرابع. أما البنى المدروسة في هذا الباب، فهي: الأشجار المُعشّمة، والكومات القابلة للدمج، وكومات فيوناتشي، وبنيّ للمعطيات التّقرّبية المعروفة باسم شجرة van Emde Boas. وأخيرًا يعالج هذا الباب بنيّ معطياتٍ خاصة تُعرف بالمجموعات المنفصلة.

ويضمُّ الجزء الثاني بابين وأربعة ملاحق:

- يعالج الباب السادس البيانات graphs وأهمُّ ما يتعلق بها من خوارزميات لها تطبيقات واسعة في علوم الحاسوب والاتصالات. فيعرض بعمق خوارزميات البحث داخل بيانٍ مع تطبيقات لها، وخوارزميات البحث عن أشجار المسح الصغرى، وخوارزميات أقصر المسارات بأغاطها المختلفة. وينتهي الباب بدراسة خوارزميات تتعلق بحساب التدفق الأعظمي داخل بيان.

- يجمع الباب السابع - وهو الأخير - مجموعة من المواضيع المختارة التي لا تقل أهمية عن مواضيع الأبواب السابقة، ولكنها تُقدِّم أكثر تقنيةً أو تعقيداً. تتميز فصولُ هذا الباب بأنها - في جلِّها - مستقلةٌ بعضها عن بعض، وتعالج مسائلَ عدَّة تُعبد منها المتخصصون في مجالاتٍ مختلفة؛ فهناك مثلاً فصلٌ يُقْنى بالحوسبة التفرعية المتعدِّدة النياسب، وآخر بمطابقة متواليات الحارف. وهناك فصولٌ تعالج مواضيعٍ وبني رياضية واسعة التطبيقات مثل: المصفوفات، ونحويلات فوريه، وخوارزميات نظرية الأعداد ذات الأهمية الكبيرة في مجال التعمية، وخوارزميات الهندسة المخوسية. ويضمُّ الباب أيضاً عدَّة فصولٍ تُثَقِّى بحلِّ مسائلٍ الأمثلة optimization مثل: البرمجة الخطية، ونظرية تعقيد المسائل، وخوارزميات التقريب.

- تقدِّم الملاحقُ تذكرةً لكلِّ ما يحتاج إليه القارئ من معارف ذات طابعٍ رياضيٍّ، مثل: مجاميع السلاسل العددية، والبني الرياضية مثل: المجموعات، والدوال، والعلاقات، والبيانات، والمصفوفات. وتذكرُ أيضاً بطرائق العد، ونظرية الاحتمالات.

ولكي يُسهِّل على القارئ الاستفادة من هذا الكتاب القيم على أكمل وجه، سيؤوِّد جزؤه الثاني بفهرسٍ ومُسَرِّدٍ ألفبائيٍّ بالمصطلحات العربية ومقابلاتها الإنكليزية، وبمسَرِّدٍ آخر بالمصطلحات الإنكليزية ومقابلاتها العربية.

نرجو أن نكون قد قدَّمنا بعلنا هذا الفائدة للقارئ العربي، وأسهمنا في تطوير وطننا وأمتنا العربية. ونأسف لما يمكن أن يكون قد غاب عنَّا من هفواتٍ وأعطاء، ونشكر سلفاً كلَّ من يُثَبِّتها عليها لتلافياها في طبعةٍ قادمة.

والله وليُّ التوفيق.

فريق التعريب



## مقدمة

وُجِدت الخوارزميات قبل وجود الحواسيب. واليوم ومع وجود الحواسيب، أصبح لدينا المزيد من الخوارزميات، وأصبحت الخوارزميات من صميم الحوسبة.

يقدم هذا الكتاب مدخلاً شاملاً لدراسة الخوارزميات الحاسوبية الحديثة، فهو يعرض عدة خوارزميات ويشملها بعمق، ومع ذلك فهو يُقَيِّص تصميمها وتحليلها في متناول القراء على اختلاف مستوياتهم. حاولنا إبقاء الشروح بسيطة دون التضحية بعمق الشمول أو بالدقة الرياضية.

يعرض كل فصل خوارزمية، أو تقنية تصميم، أو مجالاً تطبيقياً، أو موضوعاً ذا صلة. تُشرِّح الخوارزميات بالعربية وبشبه رماز مصمَّم بحيث يتمكن أي شخص ملِّم بالبرمجة من قراءته. يتضمن هذا الكتاب 244 شكلاً — يحتوي العديد منها على عدة أجزاء — توضح كيف تعمل الخوارزميات. ولما كنا نشدد على التفاعلية باعتبارها معياراً تصميمياً، فقد ضمنا تحليلات دقيقة لأزمان تنفيذ جميع خوارزمياتنا.

هذا النص معدُّ أصلاً للاستعمال في المقررات المتعلقة بالخوارزميات وبنى المعطيات في المرحلة الجامعية وفي الدراسات العليا. ولما كان هذا الكتاب يناقش القضايا الهندسية في تصميم الخوارزميات، إضافةً إلى الجوانب الرياضية، فهو ملائم أيضاً لتعلُّم المختصين التقنيين ذاتياً.

في هذه الإصدار الثالث، قمنا بتحديث كامل الكتاب مرة أخرى. تشمل هذه التغييرات طبعاً واسماً، من إدراج فصول جديدة، وتعديل لشبه الرماز، واستخدام أسلوب كتابة موجه للقارئ.

إلى المدرِّس:

لقد صمَّمنا هذا الكتاب ليكون متعدد الجوانب وكاملاً في الوقت نفسه. ستجده مفيداً لمقررات متنوعة، ابتداءً من مقررات في بنى المعطيات في المرحلة الجامعية وحتى دروس في الخوارزميات للدراسات العليا. ولما كنا قد قدمنا مواد أكثر بكثير مما يمكن أن يستوعبه مقرر نموذجي في فصل واحد، فيمكنك أن تعتبر هذا الكتاب مائدة مفتوحة متنوعة يمكنك أن تنتقي وتختار منها أفضل مادة تدعم المقرر الذي ترغب في تدريسه.

ستجد أنه من السهل تنظيم مقرر المتعلق بالفصول التي نحتاج إليها فقط، فقد جعلنا الفصول مستقلاً بعضها عن بعض نسبياً، فلا تفلق بشأن الارتباط غير المتوقع وغير الضروري لفصل بآخر. يعرض كل فصل

لمادة بتدرج من الأسهل إلى الأصعب، وتشير حدود المقاطع إلى نقاط التوقف الطبيعية. قد تستخدم في أحد مقررات المرحلة الجامعية، المقاطع الأولى من الفصل، في حين قد يغطي مقرر في الدراسات العليا الفصل بأكمله. ضمناً في الكتاب 957 تمريناً و 158 مسألة. ينتهي كل مقطع بتمارين، وكل فصل بمسائل. تكون التمارين عادة أسئلة قصيرة تختبر تمكن الطالب من المادة. بعض هذه التمارين بسيط للتحقق الذاتي من الأفكار، في حين أن بعضها الآخر أكثر عمقاً ويناسب وظيفة مثلية. تمثل المسائل دراسة حالة أكثر تفصيلاً، وغالباً ما تقدم مادة جديدة؛ وتآلف غالباً من عدة أسئلة تقود الطالب عبر الخطوات اللازمة للوصول إلى حل.

انطلاقاً من خبرتنا في إصدارات سابقة من هذا الكتاب، وضعنا حلولاً لبعض المسائل والتمارين، لا لكليها، وجعلناها متاحة للعموم. يشير موقعنا <http://mitpress.mit.edu/algorithms/> إلى هذه الحلول. من المناسب أن نتفقد هذا الموقع لتحقيق من عدم تضمنه حلولاً لتمرين أو مسألة تخطط لإعطائها وظيفة مثلية. وتوقع أن نكرر مجموعة الحلول التي نُحلّها على مر الوقت، لذلك قد نحتاج إلى تفقد الموقع في كل مرة تدرس المادة. وضعنا علامة (\*) على المقاطع والتمارين التي تناسب طلاب الدراسات العليا أكثر من المرحلة الجامعية. ولا يعني وضع علامة النجمة على مقطع بأنه أكثر صعوبة من المقطع الذي ليس موسوماً بنجمة، ولكنه قد يتطلب فهماً رياضيات أكثر تقدماً. كذلك، فقد تتطلب التمارين الموسومة بالنجمة خلفية أكثر عمقاً، أو إبداعاً أكثر من الوسطي.

### إلى الطالب:

نأمل أن يزدك هذا الكتاب بمدخل ممتع في مجال الخوارزميات. حاولنا جعل جميع الخوارزميات مفهومة ومثيرة للاهتمام. لمساعدتك عندما تصادفك خوارزميات غير شائعة أو صعبة، وصنّفنا كل واحدة منها خطوة بخطوة، وقدمنا أيضاً شرحاً دقيقاً للرياضيات الضرورية لفهم تحليل الخوارزميات. إذا كانت لديك معرفة بسيطة عن موضوع ما، ستجد الفصول منظمة بحيث يمكنك تصفح المقاطع التمهيدية والبدء سريعاً بمواد أكثر تقدماً. إن هذا الكتاب كبير، وسيغطي صفك على الأرجح جزءاً من موادك فقط. غير أننا حاولنا أن نجعله كتاباً مفيداً لك الآن ككتاب مرجعي للمقرر، ولاحقاً في مهنتك كمرجع مكتبي رياضي أو كدليل هندسي.

ما هي للتطلبات التي تلزمك لقراءة هذا الكتاب؟

- أن تكون لديك بعض الخبرة البرمجية. وبالتحديد، يجب أن تكون قد استوعبت الإجراءات العودية recursive procedures، وبنى للمعطيات البسيطة كالصفحة array، واللوائح المترابطة linked lists.
- أن تكون لديك بعض الإلمام فيما يتعلق بالبراهين الرياضية، وخاصة البراهين بالاستقراء الرياضي mathematical induction. نعتد بعض أجزاء هذا الكتاب على بعض المعارف بحسابات التكامل الأولية. فيما عدا ذلك، يُعلمك البابان I و VIII من هذا الكتاب جميع التقانات الرياضية التي ستحتاج إليها.

لقد سمعنا طلبكم الواضح والصريح لتزويدكم بحلول المسائل والتمارين. بضع موقعنا <http://mitpress.mit.edu/algorithms/> وصلات إلى حلول لبعض المسائل والتمارين. يمكنكم متى شئتم مقارنة حلولكم بحلولنا، لكن لا ترسلوا إلينا حلولكم.

### إلى المختص:

إن الطيف الواسع للمواضيع الموجودة في هذا الكتاب يجعل منه دليلاً ممتازاً عن الخوارزميات. ولما كان كل فصل مستقل المضمون تقريباً، يمكنك أن تركز على أكثر المواضيع أهمية بالنسبة إليك. إن أغلب الخوارزميات التي نتطرق إليها ذات فائدة عملية عظيمة. لذلك، نحن نناقش قضايا التحفيز ومسائل هندسية أخرى. نقدم غالباً بدائل عملية لبعض الخوارزميات التي لها أهمية نظرية في المقام الأول. إذا رغبت في تنفيذ أي من الخوارزميات، سنجد أن ترجمة شبه الرمز pseudocode الذي كتبناه، إلى لغة البرمجة المفضلة لديك، هي مهمة مباشرة إلى حد ما. لقد صممنا شبه الرمز لعرض كل خوارزمية عرضاً واضحاً وموجزاً. ومن ثم، نحن لا نناقش قضايا معالجة الخطأ وقضايا هندسة البرمجيات الأخرى التي تتطلب افتراضات محددة حول البيئة البرمجية التي تستخدمها. نحاول عرض كل خوارزمية عرضاً بسيطاً ومباشراً دون أن نسمح لخصوصيات لغة برمجة محددة أن تغطي جوهرها.

نحن نفهم أنك إذا كنت تستخدم هذا الكتاب خارج نطاق أي مقر، فربما لن تكون قادراً على تدقيق حلولك للمسائل والتمارين ومقارنتها بالحلول التي قد يقدمها مدرس. يُوجد على موقعنا <http://mitpress.mit.edu/algorithms/> وصلات إلى حلول لبعض المسائل والتمارين، بحيث تتمكن من تدقيق عملك. يرجى عدم إرسال حلولكم لنا.

### إلى زملائنا:

لقد وفرنا مراجع ومؤشرات شاملة على الأدبيات الحالية. ينتهي كل فصل بمجموعة من الملاحظات التي تقدم تفاصيل تاريخية ومراجع. غير أن ملاحظات الفصول لا تقدم دراسة مرجعية كاملة لمجال الخوارزميات كله. وقد يصعب التصديق أن قيود حجم الكتاب منعنا من إضافة خوارزميات هامة عديدة.

رغم الأعداد الضخمة لطلبات الطلاب للحصول على حلول للمسائل والتمارين، فقد اخترنا سياسة عدم التزويد بمراجع لحل المسائل والتمارين، لتلا تسول للطلاب أنفسهم البحث عن حلٍّ للمسألة بدلاً من حلها بأنفسهم.

### التغييرات على الإصدار الثالث:

ما الذي تغير بين الإصدار الثاني والثالث من هذا الكتاب؟ يعادل حجم التغييرات الحالية حجم التغييرات بين الإصدارين الأول والثاني. وكما ذكرنا عن التغييرات في الإصدار الثاني، قد نجد التغييرات في هذا الإصدار كثيرة



أو محدودة تبعاً لكيفية قراءتك للكتاب.

تبيّن نظرة سريعة إلى الفهرس أن معظم فصول ومقاطع الإصدار الثاني موجود في الإصدار الثالث. قمنا بحذف فصلين ومقطع واحد، ولكننا أضفنا ثلاثة فصول جديدة ومقطعين، علماً الموجود في هذه الفصول الجديدة.

لقد حافظنا على التنظيم المعلن للمعتمد في الإصدارين السابقين. بدلاً من تنظيم الفصول وفق مواضيع (أو مجالات تطبيق) للمسائل فقط أو وفق التقنيات فقط، يعتمد الكتاب مزيجاً من الاثنين معاً، فهو يتضمن فصولاً تعتمد التقنيات، مثل *فرق-تشد*، *divide-and-conquer*، والبرمجة الديناميكية *dynamic programming*، والخوارزميات الشرهة *greedy algorithms*، والتحليل المخثد *amortized analysis*، وتعقيد المسائل *NP-Completeness*، وخوارزميات التقريب *approximation algorithms*. لكنه يتضمن أيضاً أجزاء كاملة عن الفرز، وبقي للمعطيات اللازمة للمجموعات الديناميكية، والخوارزميات المتعلقة بمسائل البيان *graph problems*. ونحن نعلم أنه على الرغم من أنك بحاجة إلى معرفة كيفية تطبيق التقنيات في التصميم والتحليل الخوارزمي، إلا أنه قلما تملك المسائل على أكثر التقنيات طوعةً لحلها.

نقدم فيما يلي ملخصاً لأهم التغييرات في الإصدار الثالث:

- أضفنا فصولاً جديدة عن أشجار *van Emde Boas*، والخوارزميات المتعددة النياسب *multithreaded algorithms*، وفصلنا المادة المتعلقة بأساسيات المصفوفات في فصلٍ ملحقٍ خاص.
- راجعنا الفصل المتعلق بالعودية *recurrences* ليشمل بصورة أوسع تقنية "فرق-تشد"، وبحيث تُطبق هذه التقنية لحل مسألتين في أول مقطعين فيه. عرض للمقطع الثاني من هذا الفصل خوارزمية *Strassen* لإيجاد جداء المصفوفات، نقلناها من الفصل المتعلق بعمليات المصفوفات.
- حذفنا فصلين كانا نادراً ما يدرّسان: الكومات الثنائية *binomial heaps*، وشبكات الفرز *sorting networks*. تظهر في هذا الإصدار فكرة أساسية من فصل شبكات الفرز، وهي مبدأ 0-1، وذلك ضمن المسألة 7-8 على شكل نوظة الفرز 0-1 في خوارزميات قارن-بدل *compare-exchange*. ولم تُعد معالجة كومات فيوناتشي *Fibonacci* تعتمد على الكومات الثنائية باعتبارها متطلباً سابقاً.
- راجعنا طريقة معالجتنا للبرمجة الديناميكية والخوارزميات الشرهة. تبدأ البرمجة الديناميكية الآن بمسألة أكثر إمتاعاً، وهي تقطيع القضبان *rod cutting*، بدلاً من مسألة جدولة خط التجميع *assembly line* الموجودة في الإصدار الثاني. إضافة إلى ذلك، ركزنا على الاستدكار أكثر مما فعلنا في الإصدار الثاني، وقدمنا فكرة بيان المسألة الجزئية على أنها طريقة لفهم زمن تنفيذ خوارزمية البرمجة الديناميكية. في مثالنا الافتتاحي عن الخوارزميات الشرهة، وهو مسألة اختيار النشاط *activity-selection*، نصل إلى الخوارزمية الشرهة مباشرة بطريقة أفضل مما كانت في الإصدار الثاني.

- تتضمن الطريقة التي نحذف ونفقا الآن عقدة من أشجار البحث الثنائية (التي تتضمن الأشجار الحمراء-السوداء) أن العقدة التي يطلب حذفها هي العقدة المخلوقة فعليًا. في الإصدارين السابقين، وفي بعض الحالات، كان من الممكن أن نحذف عقدة ماء، وثقل محتوياتها إلى العقدة التي تمزج إلى إجراء الحذف. مع طريقتنا الجديدة لحذف العقد، إذا كانت هناك مكونات أخرى من البرنامج تحافظ على مؤشرات إلى عقد في الشجرة - فلن ينتهي بها الأمر مع مؤشرات قديمة إلى عقدٍ حُذفت.
- إن المادة المتعلقة بشبكات التدفق تجعل التدفق الآن معتمدًا كليًا على الوصلات. إن هذه المقاربة أكثر بداهة من التدفق الشبكي net flow المستخدم في الإصدارين السابقين.
- أصبح الفصل المتعلق بعمليات المصفوفات أصغر مما كان عليه في الإصدار الثاني نتيجة نقل المادة المتعلقة بأساسيات للمصفوفات وخوارزمية Strassen إلى فصول أخرى.
- عدّلنا طريقة معالجة خوارزمية Knuth-Morris-Pratt لمطابقة المتتاليات المخفية.
- صححنا عدة أخطاء، نشرنا معظمها على موقع الوب الخاص بتصحيح أخطاء الإصدار الثاني، وبقي بعضها دون نشر.
- اعتمادًا على العديد من الطلبات، غرّنا التركيب النحوي لشبه الرماز عما كان عليه. نستخدم الآن "==" للتعبير عن الإسناد assignment، و "=" لاعتبار المساواة، كما في C و C++ و Java و Python. وحذفنا، كذلك، الكلمات المفتاحية do و then واصطلحنا على "/" باعتباره رمزًا للتعليقات الممتدة حتى نهاية السطر. نستخدم الآن أيضًا التدوين النقطي dot-notation للدلالة على واصفات الغرض object attributes. يبقى شبه الرماز إجرائيًا، وليس غرضي التوجه. وبعبارة أخرى، بدلاً من تنفيذ الطرائق على الأغراض، نستخدم الإجراءات ببساطة مع تمرير الأغراض باعتبارها موسطات.
- أضفنا 100 تمرين جديد و 28 مسألة جديدة. كذلك حدّثنا العديد من المراجع وأضفنا عدة مراجع جديدة.
- في النهاية، راجعنا الكتاب بكامله، وأعدنا صياغة الجمل، والفقرات، وللقاطع لجعل الكتابة أكثر وضوحًا وفعالية.

### موقع الوب:

يمكنك استخدام موقع الوب الخاص بالكتاب <http://mitpress.mit.edu/algorithms/> للحصول على معلومات إضافية وللتواصل معنا. يتضمن موقع الوب وصلات إلى لائحة الأخطاء المعروفة، وإلى حلول لتمرينين ومسائل مختارة، و(بالطبع) إلى لائحة تشرح نكات الأساتذة السخيفة، إضافة إلى محتويات أخرى قد نضيفها. يدلكم موقع الوب أيضًا على كيفية الإبلاغ عن الأخطاء أو التزويد بالمقترحات.

## كيف أنتجت هذا الكتاب:

أنتج الإصدار الثالث، مثل الإصدار الثاني، باستخدام LATEX2 $\epsilon$ . استخدمنا البنية Times مع مجموعة أنماط رياضية باستخدام البنية MathTime Pro 2. نشكر Michael Spivak من شركة Publish or Perish، و Lance Cames من شركة Personal Tex، و Tim Tregubov من كلية Dartmouth College، وللدعم التقني. جميعاً - كما في الإصدارين السابقين - دليل للمصطلحات index باستخدام Windex، وهو برنامج كيند بلغة C، وجرى إنتاج المراجع باستخدام BIBTEX. وأنشأنا ملفات PDF لهذا الكتاب على حاسوب MacBook نظام تشغيله OS 10.5.

رسمنا الرسوم التوضيحية للإصدار الثالث باستخدام MacDraw Pro، حيث وضعت بعض التعابير الرياضية في الرسوم التوضيحية باستخدام حزمة psfrag المخصصة لـ LATEX2 $\epsilon$ . لسوء الحظ، كانت MacDraw Pro برمجية قديمة، ولم تعد تتوافق منذ عقد. غير أنه لحسن الحظ، كانت لدينا بعض حواسيب الماكنتوش التي تشتغل ضمن بيئة ماکنتوش كلاسيك بنظام تشغيل OS 10.4، وبذلك يمكننا تشغيل MacDraw Pro. وقد وجدنا أن استخدام MacDraw Pro ضمن بيئة كلاسيك، أسهل بكثير من أية برمجية رسم أخرى لأنماط الرسومات التي تصاحب عادة النصوص في علوم الحاسوب، وهي تنتج خرجاً جميلاً<sup>1</sup> من يدري حتى متى مستمر حواسيبنا ماکنتوش (من عهد ما قبل Intel) بالعمل، لذلك إذا كان هناك من يسمنا من شركة Apple: "رجاءاً احسنوا نسخة من MacDraw Pro تتوافق مع نظم التشغيل الأخرى."

## شكر خاص بالإصدار الثالث:

نحن نعمل مع مطبعة MIT Press منذ ما يزيد على عقدين حتى الآن، وبالمثل من علاقة ممتازة! نحن نشكر Ellen Farn، و Bob Prior، و Ada Brumstein، و Mary Reilly لمساعدتهم ودعمهم. لقد كنا متفاعلين جغرافياً أثناء إنتاج الإصدار الثالث، حيث كنا نعمل في قسم علوم الحاسوب في كلية Dartmouth، وفي مختبر علوم الحاسوب والذكاء الصناعي في MIT، وقسم الهندسة الصناعية وبحوث العمليات في جامعة Columbia، ونحن نشكر جامعتنا تلك وزملائنا لتوفيرهم بيئات عمل محفزة وداعمة. مرة أخرى، أشكركم Julie Sussman، من جمعية P.P.A، بفضل جهودها في التصحيح قبل الطبع؛ فقد

<sup>1</sup> لقد جربنا عدة برامج رسم تشتغل ضمن بيئة Mac OS X، ولكن كان لكل منها نقائص مقارنةً ببرنامج MacDraw Pro. لايجاز، حاولنا إنتاج رسوم هذا الكتاب التوضيحية باستخدام برنامج رسم آخر مشهور جداً، ولكننا وجدنا بأنه استغرق خمسة أضعاف الوقت، على الأقل، مقارنةً ببرنامج MacDraw Pro لإنتاج كل رسم توضيحي، ولم تكن الرسوم بالجودة نفسها. بناءً على ذلك، كان قرارنا بالتحويل إلى تشغيل MacDraw Pro على حواسيب ماکنتوش قديمة.

ذهلنا مراراً من الأخطاء التي فأتنا وكشفتها Julie. وساعدتنا Julie أيضاً على تحسين عرضنا في أماكن عديدة، ولو كان هناك مسابقة لانتخاب مشاهير في مجال التحرير التقني، فلا ريب أنها ستكون أول من ينتخب. فهي مدهشة! شكراً شكراً يا Julie! كذلك عثر Priya Natarajan على بعض الأخطاء التي استطعنا تصحيحها قبل إرسال الكتاب إلى المطبعة. فإن بقيت أية أخطاء (وحنما، لازال هناك البعض) فهي مسؤولية المؤلفين (ربما أضيفت بعد أن قرأت Julie مادة الكتاب).

استقيث معالجة أشعار van Emde Boas من مسودات Erik Demaine، التي تأثرت بدورها بـ Michael Bender. وقد أضفنا في هذا الإصدار أيضاً أفكاراً من Bradley Kuszmaul و Javed Aslam و Hui Zha.

اعتمد الفصل المتعلق بتعدد النياسب على مسودات كتبت أصلاً بالمشاركة مع Harald Prokop. تأثرت مادة الكتاب بعدة أعمال أخرى ضمن مشروع Cilk في MIT، ومنها أعمال Bradley Kuszmaul و Matteo Frigo. واستوحي تصميم شبه الرماز المتعدد النياسب من توسيعات Cilk الخاصة بمعهد MIT للغة C، ومن توسيعات Cilk++ الخاصة بشركة Cilk Arts للغة C++.

نشكر أيضاً العديد من قراء الإصدارين الأول والثاني الذين بلغوا عن أخطاء أو قدموا اقتراحات لتحسين هذا الكتاب. وقد صغحننا جميع الأخطاء الفعلية التي جرى التبليغ عنها، وضمننا ما استطعنا من الاقتراحات. ونحن سعداء بأن عدد هؤلاء المساهمين أصبح كبيراً إلى درجة يتعذر تعداد أسمائهم جميعاً، وهذا ما نأسف بشأنه.

في النهاية، نشكر زوجاتنا: Rebecca و Gail Rivest و Wendy Leiserson و Nicole Cormen و أولادنا: Ivy و Ricky و Will و Debby و Katie Leiserson و Alex، و Christopher Rivest و Molly و Noah، و Benjamin Stein لحيهم ودعمهم لنا خلال مدة تحضيرنا لهذا الكتاب. لقد ساهم صبرهم وتشجيعهم في جعل هذا المشروع ممكناً. نحن نقديهم هذا الكتاب مع حبنا.

Lebanon, New Hampshire	من	THOMAS H. CORMEN
Cambridge, Massachusetts	من	CHARLES E. LEISERSON
Cambridge, Massachusetts	من	RONALD L. RIVEST
New York, New York	من	CLIFFORD STEIN



---

مَدْخُلُ إِلَى الْخَوَارِزْمِيَّاتِ

الإصدار الثالث



## تمهيد

إن هذا الباب من الكتاب سيحملك تبدأ بالتفكير في تصميم الخوارزميات وتحليلها؛ فقد أُعدَّ ليكون مقدمة سهلة في كيفية توصيف الخوارزميات، ومقدمة لبعض استراتيجيات التصميم التي سنستخدمها في الكتاب، وللكثير من الأفكار الأساسية المستخدمة في تحليل الخوارزميات. وستبني الأجزاء التالية على هذه القاعدة. يعطى الفصل الأول نظرة شاملة عن الخوارزميات وموقعها في النظم الحاسوبية الحديثة؛ فهو يُعرِّف الخوارزمية ويسرد بعض الأمثلة. ويبيِّن كذلك أننا سنعتبر الخوارزميات تقانة، تمامًا كالبنية المادية السريعة، وواجهات المستخدم البيانية، والنظم الغرضية التوجه، والشبكات.

سنصادف في الفصل الثاني خوارزمياتنا الأولى التي تحلُّ مسألة فرز متتالية من  $n$  عددًا. هذه الخوارزميات مكتوبة بشبه رماز pseudocode، غير قابل للترجمة مباشرة إلى أية لغة برمجة مألوفة، إلا أنه ينقل بنية الخوارزمية بوضوح كافٍ لمُكنِّكك من تنفيذها بلغة البرمجة التي تختارها. إن خوارزميات الفرز التي سندرسها هي خوارزمية الفرز بالإدراج insertion sort، التي نتمدُّ مبدأً تدرجيًّا incremental approach، وخوارزمية الفرز بالدمج merge sort التي نستخدم تقنيةً عَوْدِيَّةً تُعرف باسم "فرق-تحد" divide-and-conquer. وسنلاحظ من هذه الدراسة أنه على الرغم من ازدياد زمن التنفيذ اللازم لكلتا الخوارزميتين مع ازدياد حجم المسألة  $n$ ، فإن معدل الزيادة يختلف في كل منهما. وسنحدِّد في هذا الفصل أزمان التنفيذ running times هذه، وسنطوِّر طريقة مفيدة لتدوين هذه الأزمان للتعبير عنها.

يُعرِّف الفصل الثالث هذا التدوينَ تعريفيًا دقيقًا، والذي نسميه التدوين للمقارب asymptotic notation. يبدأ الفصل بتعريف عدة تدوينات مقارنة، نستخدمها لجِدُّ أزمان تنفيذ الخوارزمية من الأعلى و/أو من الأسفل. أما بقية الفصل الثالث، فهي في المقام الأول عرضٌ للتدوين الرياضي mathematical notation، الغرض منه التأكد أن استخدامك للتدوين يطابق طريقة التدوين المُعتمدة في هذا الكتاب، وليس مجرد أن تتعلَّم أفكارًا رياضية جديدة.



يقوم الفصل الرابع بعَمَي أكبر في طريقة "فَرْق-تَشَدُّ" التي تم التطرق إليها في الفصل الثاني. ويوفر أمثلة إضافية على خوارزمياتنا، تشمل طريقة Strassen للمهشة لضرب مصفوفتين مربعيتين. ويحتوي هذا الفصل أيضًا على طرائق لحل العلاقات المتّوَدِيّة، وهي مفيدة في وصف أزمّة تنفيذ الخوارزميات القوَدِيّة. إحدى التقنيات الفعّالة هي "الطريقة الرئيسة master method" التي سنستخدمها غالبًا لحل العلاقات القوَدِيّة التي ترد في خوارزميات فَرْق-تَشَدُّ. ومع أن معظم الفصل الرابع مَحْصَصٌ لإثبات صحة "الطريقة الرئيسة"، إلا أنه يُمكنك الآن تجاوز هذا البرهان وأن تظّل تستخدم "الطريقة الرئيسة".

يعرض الفصل الخامس التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة. نستخدم عادةً التحليل الاحتمالي لتحديد زمن تنفيذ خوارزمية ما في الحالات التي قد يختلف فيها زمن تنفيذ الخوارزمية لمُدخَلات مختلفة لها الحجم نفسه، وذلك بسبب وجود توزيع احتمالي أصيل في جوهر المسألة. في بعض الحالات نفترض أن قيم الدخل تتبع توزيعًا احتماليًا معروفًا، وهذا ما يسمح لنا بحساب متوسط زمن التنفيذ على جميع المُدخَلات الممكنة. في حالات أخرى، لا يأتي التوزيع الاحتمالي من قيم الدخل بل من الخيارات العشوائية التي تُتَّخَذُ في سياق الخوارزمية. أما الخوارزمية التي لا يَتَحَدَدُ سلوكها من اختلاف قيم الدخل فقط، بل بالاعتماد على قيم يُولَّدُها مولّد أَعْدَادٍ عشوائية، فسمى خوارزمية ذات "عشوائية مضافة". نستطيع استخدام خوارزميات ذات عشوائية مضافة لِحَلِّ المُدخَلات تتبع توزيعًا احتماليًا - وهكذا نضمن عدم وجود دخل خاص ينسب دائماً بأداء سيئ - أو لتعَيّن حدود معدل خطأ الخوارزميات التي يسمح لها بإعطاء نتائج خاطئة بحدود محدد.

تحتوي للملاحق (أ)-(ت) مادة رياضية إضافية سنجدها ذات فائدة عندما تقرأ هذا الكتاب. ومن المرجح أنك عانيت جزئيًا كبيرًا من محتوى فصول الملحق قبل قراءتك لهذا الكتاب (علماً بأن التعريفات والمصطلحات التلويحية الخاصة التي نستخدمها قد تختلف أحيانًا عما رأيته سابقًا)، لذا ينبغي أن نتعامل مع الملاحق على أنها مادة مرجعية نعوّذُ إليها عند الحاجة. من ناحية أخرى، يُحتمل أنك لم تطلّع من قبل على معظم محتوى الباب I؛ فجميع فصول هذا الباب والملاحق مكتوبة بأسلوب تعليمي مبسط.

# 1 دور الخوارزميات في الحوسبة

ما هي الخوارزميات؟ وما أهمية دراستها؟ وما دورها بالنسبة إلى بقية التقانات المستخدمة في الحواسيب؟ سنحيط في هذا الفصل عن هذه الأسئلة.

## 1.1 الخوارزميات

الخوارزمية *algorithm* عموماً هي أي إجراء مُحَوَّس مُعرَّف جيداً، يأخذ قيمة أو مجموعة قيم نسميها الدخول *input*، ويُنتج قيمة أو مجموعة قيم نسميها المخرج *output*. فالخوارزمية إذاً هي متالية محدودة من الخطوات المُحَوَّسة تُحوِّل الدخول إلى مخرج.

يمكن أن ننظر إلى الخوارزمية على أنها أداة لحل مسألة مُحَوَّسة *computational problem* مُوصَّفة جيداً. يُعَيَّف نص للمسألة العلاقة المطلوبة بين الدخول والمخرج باستخدام مصطلحات عامة. نصف الخوارزمية إجراءً مُحَوَّساً محدداً لتحقيق علاقة الدخول/المخرج تلك.

مثلاً قد نحتاج إلى فرز متالية من الأعداد حسب ترتيب غير متناقص. يتكرر ظهور هذه المسألة في الواقع العملي وتُوفّر التربة الخصبة لعرض كثير من أدوات التحليل وتقانات التصميم القياسية، وفيما يلي عرض لكيفية تعريف مسألة الفرز *sorting problem* تعريفاً صورياً:

الدخول: متالية من  $n$  عدداً  $(a_1, a_2, \dots, a_n)$ .

المخرج: تبديل (إعادة ترتيب)  $(a'_1, a'_2, \dots, a'_n)$  متالية الدخول بحيث يكون  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

مثال: ليكن لدينا متالية الدخول  $(31, 41, 59, 26, 41, 58)$ ، نُنتِج خوارزمية ما للفرز في عرجها المتتالية  $(26, 31, 41, 41, 58, 59)$ . تسمى متالية الدخول للماتلة للمتتالية السابقة مُنتَسَخ *instance* مسألة الفرز. يتكون مُنتَسَخ مسألة ما *instance of a problem* عموماً من الدخول اللازم لحساب حل هذه المسألة (وُتحقق أية شروط مفروضة في نص للمسألة).

يُعدُّ الفرزُ عمليةً أساسيةً في علم الحاسوب، لأن برامج عديدة تُستخدِمُها باعتبارها خطوةً مرحلية.

ونتيجة لذلك، لدينا عدد كبير من خوارزميات الفرز الجيدة. ويعتمد اختيار الخوارزمية الفضلى لتطبيقي محدد على عدد من العوامل، منها: عدد العناصر التي ينبغي فرزها، ومدى ترتيب هذه العناصر سلفاً، والقيود الملزمة على قيم العناصر، ومعاملة الحاسوب، ونوع تجهيز التخزين المستخدمة: ذاكرة رئيسة، أو أقراص، أو أشرطة مغناطيسية.

نقول عن خوارزمية ما إنها **صحيحة** *correct* إذا توقفت عند الخرج الصحيح لكل متسوخ دخل، ونقول إن الخوارزمية **الصحيحة** *تحل* *solves* المسألة **المُحسوبة** *المعطاة*. أما الخوارزمية **الخاطئة**، فقد لا تتوقف على الإطلاق عند بعض متسوخات الدخل، أو قد تتوقف عند جواب غير صحيح. وعلى عكس ما قد تتوقعه، قد تكون بعض الخوارزميات **الخاطئة** مفيدة في بعض الحالات، إذا أمكننا التحكم في معدل خطئها. سنرى مثلاً على خوارزمية يمكن التحكم في معدل خطئها في الفصل 3! عندما ندرس الخوارزميات المستخدمة لإيجاد أعداد أولية كبيرة. إلا أننا سوف نتم في الحالة العامة بالخوارزميات **الصحيحة** فقط.

يمكن أن تُوصَف الخوارزمية باللغة الإنكليزية، كبرنامج حاسوبي، أو حتى كتصميم مادي. الشرط الوحيد أن يُقدَّم الوصف وصفاً دقيقاً للإجراء المُخَوَّص الواجب اتباعه.

### ما هي أنواع المسائل التي يمكن حلها باستخدام الخوارزميات؟

لبست مسألة الفرز هي المسألة **المُحسوبة** الوحيدة التي طُوِّرت من أجلها الخوارزميات (قد يكون ذلك قد ورد في ذهنك عندما رأيت حجم هذا الكتاب)، إذ تشمل التطبيقات العملية للخوارزميات كل المجالات ومنها الأمثلة التالية:

- حقق مشروع الجينوم البشري Human Genome Project تقدماً كبيراً باتجاه أهداف تعريف كل الـ 100,000 جين الموجودة في الحمض النووي البشري DNA، وتحديد متواليات 3 مليارات من الأزواج الكيميائية الأساسية التي تُكوِّن الحمض النووي البشري، وتخزين هذه المعلومات في قاعدة معطيات، وتطوير أدوات لتحليل هذه المعطيات. تتطلب كل من هذه الخطوات خوارزميات معقدة. ومع أن حلول المشاكل المختلفة التي تشملها هذه الخطوات تقع خارج نطاق هذا الكتاب، إلا أن كثيراً من طرائق حل هذه المسائل الحيوية نستخدم أفكاراً من عدة فصول في هذا الكتاب، وهكذا تُتيح للعلماء تحقيق مهمات باستخدام الموارد المتاحة بفعالية. يؤدي استخدام خوارزميات فعالة إلى توفير في وقت الإنسان وفي وقت الآلة، وفي المال عندما يمكن استنتاج معلومات أكثر من التقنيات المخبرية.
- تتيح الإنترنت للمستخدمين في جميع أنحاء العالم نفاذاً سريعاً واستعادة كميات كبيرة من المعلومات. وبمساعدة خوارزميات ذكية، أصبحت الصفحات على الإنترنت قادرة على إدارة ومعالجة هذه الكميات الكبيرة من المعطيات. تشمل الأمثلة على المسائل التي تُستخدم الخوارزميات أساساً لها مسألة إيجاد المسارات الجيدة التي يجب أن تُنقَل عليها للمعطيات (تُظَهَر تقانات حل مثل هذه المسائل في

(الفصل 24)، ومسألة استخدام محرك بحث لإيجاد صفحات فيها معلومات معينة بسرعة (التقنيات المتعلقة بذلك موجودة في الفصل 11 والفصل 32).

■ تتيح التجارة الإلكترونية التفاوض على البضائع والخدمات وتبادلها إلكترونياً، وتعتمد على خصوصية المعلومات الشخصية مثل أرقام البطاقات الائتمانية، وكلمات السر، والكشوف المصرفية. تشمل التقانات الأساسية المستخدمة في التجارة الإلكترونية تشفير للمفتاح العام public-key cryptography والتوقيعات الرقمية digital signatures (المشروحتان في الفصل 31)، اللتان تعتمدان على الخوارزميات الرقمية numerical algorithms ونظرية الأعداد number theory.

● غالباً ما تحتاج مشاريع التصنيع والمشاريع التجارية الكبيرة الأخرى إلى تخصيص الموارد النادرة بأكثر الطرق نفعا. فقد ترغب شركة نفط في معرفة أماكن وضع الآبار ليكون ربحها المتوقع أكبر ما يمكن. وقد يرغب مرشح سياسي في أن يحدد أين ينفق الأموال في شراء الدعاية الانتخابية لزيادة فرص الكسب في الانتخابات قدر الإمكان. وقد ترغب شركة طيران في توزيع أطقمها على الرحلات بأقل الطرق كلفة، بحيث تتأكد أنها تلبى حاجة كل رحلة طيران وأنها تراعي الأنظمة الحكومية المتعلقة بمجدولة الطاقم. وقد يرغب مزود خدمة إنترنت في تحديد مكان وضع موارد إضافية لخدمة زبائنه بفاعلية أكبر. كل هذه الأمثلة هي مسائل يمكن أن تُحل باستخدام البرمجة الخطية، التي سوف نُدْرُسُها في الفصل 29.

ومع أن بعض تفاصيل هذه الأمثلة تقع خارج نطاق هذا الكتاب، إلا أننا نُقرض التقنيات الأساسية التي تنطبق على هذه المسائل وعلى مجالاتها. ستطرق أيضاً في هذا الكتاب إلى كيفية حل كثير من المسائل الخاصة، منها:

● لكن لدينا خارطة طريق مُحدّث عليها المسافة بين كل زوج من التقاطعات المتجاورة، ونرغب في تحديد أصغر طريق يصل بين تقاطعين. يمكن أن يكون عدد الطرق كبيراً جداً، حتى لو استثنينا الطرق التي تقاطع مع نفسها. كيف يمكن اختيار أقصر الطرق؟ هنا تُنمذج خارطة الطرق (التي هي نفسها نموذج للطرق الحقيقية) ككيان graph (سنجده في الباب VI والملحق ب)، ونرغب في إيجاد أقصر مسار من عقدة إلى أخرى في البيان. سوف نرى كيف يمكن حل هذه المسألة بفعالية في الفصل 24.

● لكن لدينا متاليتان مرتبتان من الرموز،  $X = \{x_1, x_2, \dots, x_m\}$  و  $Y = \{y_1, y_2, \dots, y_n\}$  ونرغب في إيجاد أطول متالية جزئية مشتركة من  $X$  و  $Y$ . إن المتالية الجزئية من  $X$  ليست إلا  $X$  كاملة أو ربما مع حذف بعض عناصرها أو ربما حذفها كلها. على سبيل المثال، إحدى المتاليات الجزئية من  $\{A, B, C, D, E, F, G\}$  هي  $\{B, C, E, G\}$ . تعطي أطول متالية جزئية مشتركة من  $X$  و  $Y$  قياساً لمدى تشابه هاتين المتاليتين. على سبيل المثال، لو كانت المتاليتان زوجين أساسيين في جدائل DNA، عندها قد نعتبرهما متشابهتين إذا كان لهما متالية جزئية مشتركة طويلة. لو احتوت  $X$  على  $m$  رموزاً و  $Y$  على  $n$

رمزاً، حينها تحتوي  $X$  على  $2^m$  و  $Y$  على  $2^n$  متتالية جزئية ممكنة، على الترتيب. إن انتخاب جميع المتتاليات الجزئية للمكعبة من  $X$  و  $Y$  ومطابقة بعضها مع بعضها الآخر يمكن أن يستغرق زمناً طويلاً جداً. الدرجة لا يمكن السماح به ما لم تكن  $m$  و  $n$  صغيرتان جداً. سنرى في الفصل 15 كيفية استخدام طريقة عامة لحل هذه المسألة بتقالب أكبر بكثير تُعرف بالبرمجة الديناميكية.

• لكن لدينا تصميم ميكانيكي متعلق بمكعبة قطع، بحيث يمكن أن تحتوي كل قطعة منتسختة من قطع أخرى، ونرغب في سرد القطع وفق ترتيب بحيث تظهر كل قطعة قبل أية قطعة تستخدمها. إذا كان التصميم مولداً من  $n$  قطعة، عندها سيكون هنالك  $n!$  من الترتيبات الممكنة، حيث تشير  $n!$  إلى دالة العامل. ولما كانت دالة العامل تنمو بسرعة أكبر من الدالة الأسية، فإننا لا نستطيع أن نؤكد عملياً كل ترتيب ممكن ثم نتحقق، ضمن ذلك الترتيب، أن كل قطعة تظهر قبل القطع التي تستخدمها (ما لم يكن لدينا سوى قطع قطع). هذه المسألة منتسخ من الغرز الطوبولوجي، وسنرى في الفصل 22 كيفية حل هذه المسألة بتقالب.

• لكن لدينا  $n$  نقطة في مستوى ونرغب في إيجاد غلاف محدب لهذه النقاط. الغلاف المحدب هو أصغر مضلع محدب يحتوي النقاط. حديثاً، يمكن أن نتخيل كل نقطة ممثلة بمسمار مغروز على لوح. يتمثل عندها الغلاف المحدب بشرط مطاطي مشدود يحيط بجميع المسامير. كل مسمار يلتف حوله الشريط المطاطي هو رأس في الغلاف المحدب. (مثلاً انظر الشكل 33.6 في الجزء الثاني من الكتاب). إن أي مجموعة من المجموعات الجزئية من النقاط وعددها  $2^n$  يمكن أن تكون رؤوس الغلاف المحدب. إن معرفة نقاط رؤوس المحدب لا يكفي، لأنه يلزمنا أيضاً معرفة ترتيب هذه النقاط. وهكذا توجد كثير من الخيارات لرؤوس الغلاف المحدب. يغطي الفصل 33 طريقتين جديتين لإيجاد الغلاف المحدب.

إن الأمثلة السابقة غيشت من قبض (كما قد تستدل من حجم هذا الكتاب) لكن هذه الأمثلة تُدرّج صفتين شائعتين تشترك فيهما كثير من المسائل الخوارزمية المثيرة للاهتمام.

1. لهذه المسائل كثير من الحلول للرشحة، الغالبية العظمى منها لا تحل المسألة. وقد يمثل إيجاد الحل الذي يحل المسألة، أو الحل الذي يمكن اعتباره "الأفضل"، تحدياً كبيراً.

2. لهذه المسائل تطبيقات عملية. أسهل الأمثلة على ذلك، من بين المسائل التي تعرضنا لها، تطبيقات مسائل أقصر المسارات. نتمم أية شركة شحن، سواء باستخدام الشاحنات أو السكك الحديدية مادياً بإيجاد أقصر المسارات عبر شبكة طرق أو شبكة سكك حديدية، لأن للمسارات الأقصر تتطلب عملاً ووقتاً أقل. أو قد تحتاج عقدة تسيير على الإنترنت إلى إيجاد أقصر مسار عبر الشبكة لتوجيه رسالة ما بسرعة. أو أن شخصاً يرغب في قيادة سيارته من نيويورك إلى بوسطن قد يريد إيجاد اتجاهات القيادة من صفحة وب مناسبة، أو أنه قد يستخدم نظام تحديد الموقع الجغرافي GPS الخاص به أثناء القيادة.

لا يوجد لكل مسألة قابلة للحل باستخدام الخوارزميات مجموعة مُعَرَّفة سهلة من الحلول المرشحة. على سبيل المثال، افترض أن لدينا مجموعة من القيم العددية تمثل عينات من إشارة، ونريد حساب تحويل فورييه المتقطع لهذه العينات. يقوم تحويل فورييه المتقطع بتحويل المجال الزمني إلى مجال ترددي، منتجة مجموعة من المعاملات العددية، بحيث يمكننا تحديد قوة مختلف الترددات في الإشارة للمأخوذة منها العينات. تمتلك تحويلات فورييه المتقطعة تطبيقات في ضغط للمعطيات وجداء كثيرات الحدود والأعداد الصحيحة الضخمة، إضافة إلى كونها تمثل نواة معالجة الإشارة. يعرض الفصل 30 خوارزمية فعالة، هي تحويل فورييه السريع (يسمى شيوعاً FFT)، لهذه المسألة، ويعرض مخططاً لتصميم دائرة الكيان الصلب لحساب تحويل فورييه السريع.

### بنى المعطيات

يحتوي هذا الكتاب أيضاً العديد من بنى للمعطيات. بنية *المعطيات Data structure* هي طريقة تخزين وتنظيم المعطيات لتسهيل الوصول إليها وإجراء تعديلات عليها. لا توجد بنية معطيات وحيدة تصلح لجميع الاحتياجات، لذلك يجب معرفة نقاط قوة وحدود استخدام الكثير من بنى المعطيات.

### أسلوب عمل

على الرغم من أنك تستطيع استخدام هذا الكتاب ككتاب وصفات "cookbook" لخوارزميات جاهزة، فقد تصادف في يوم ما مسألة لا تستطيع أن تجد لها خوارزمية جاهزة منشورة (على سبيل المثال كثير من نماين ومساائل هذا الكتاب). سوف يُقْلَمُكُ هذا الكتاب تقنيات تحليل وتصميم الخوارزميات بحيث تستطيع تطوير خوارزميات بنفسك، وترهن أنها تعطي الجواب الصحيح، وتترك مدى فعاليتها. تشرح الفصول المختلفة الأفكار المختلفة لحل المسائل الخوارزمية. وتشرح بعض الفصول مسائل خاصة، على سبيل المثال إيجاد الوسط الحسابي *median* وإحصائيات الترتيب في الفصل 9، وحساب شجرات للمسح الصغرى في الفصل 23، وتحديد التدفق الأعظمي في شبكة في الفصل 26. تشرح الفصول الأخرى تقنيات، مثل فُرْقَى-نَشْد في الفصل 4، والبرمجة الديناميكية في الفصل 15، والتحليل للمُخَمَّد *amortized analysis* في الفصل 17.

### مسائل صعبة

يعالج الجزء الأكبر من هذا الكتاب خوارزميات فعالة. مقياسنا الاعتيادي للفعالية هو السرعة، بمعنى آخر، الزمن الذي تستغرقه خوارزمية ما للوصول إلى النتيجة. مع ذلك هنالك بعض المسائل لا يُفْرُفُ لها حل فعال. يدرس الفصل 34 مجموعة جزئية هامة من هذه المسائل، تعرف باسم *NP-complete*.

لماذا تعتبر مسائل *NP-complete* هامة؟ أولاً، ومع أنه لم توجد قط أية خوارزمية فعالة لمسألة من هذا النوع، لم يُثَبِّت أحد عدم إمكانية وجود خوارزمية فعالة لهذه المسائل. وبعبارة أخرى، لا علم لنا بوجود

خوارزميات فعالة لمسائل NP-complete. ثانياً، تمتلك مجموعة مسائل NP-complete خاصية لافتة للنظر، وهي أنه إذا وجدت خوارزمية فعالة لإحداها، عندها توجد خوارزميات فعالة لجميع مسائل هذه المجموعة. إن هذه العلاقة بين مسائل NP-complete جعلت التغلب على النقص في الحلول الفعالة جميعها أكثر إثارة للتحدي. ثالثاً، تشابه بعض مسائل NP-complete، المسائل التي نعرف لها حلولاً فعالة من مسائل الخوارزميات الفعالة إلا أنها مختلفة عنها. وقد أثار اهتمام علماء الحاسوب كيف أن تغييراً بسيطاً في طرح المسألة قد يسبب تغييراً كبيراً في فعالية أفضل خوارزمية معروفة.

ينبغي أن نتعرف مسائل NP-complete لأنه من المدهش أن بعضها يبرز كثيراً في التطبيقات الواقعية. إذا ما طلب إليك ذات يوم ابتداء خوارزمية فعالة لمسألة NP-complete، فعلى الأرجح أنك ستقضي وقتاً طويلاً في بحث غير مشر. فإن استطعت إثبات أن المسألة هي NP-complete، فمن المحدي أن تصرف وقتك في تطوير خوارزمية فعالة تعطي حلاً جيداً، ولو كان ليس أفضل حلٍّ ممكن.

واليك مثلاً واقعياً: لنفترض أن شركة توزيع بضائع لها مستودع مركزي. تمثّل الشركة شاحنتها بالبضائع في المستودع المركزي وترسلها لتوزيع البضائع على عدة عناوين يومياً. يجب أن تُبَيّن الشاحنة جولة لها في نهاية اليوم وتعود إلى محطة الانطلاق لتكون جاهزة للتحميل في اليوم التالي. تريد الشركة - بمهدف خفض الكلف - اختيار ترتيب نقاط توقف كل شاحنة للتوزيع بحيث تقطع أقصر مسافة ممكنة. هذه المسألة هي "مسألة البائع الجوال" المعروفة، وهي مسألة NP-complete، وليس لها خوارزمية فعالة معروفة. ومع ذلك، نعرف وفق فرضيات محددة خوارزميات فعالة تعطي مسافة إجمالية لا تزيد كثيراً عن أصغر مسافة ممكنة. يناقش الفصل 35 "خوارزميات التقريب" هذه.

## التوازي

استطعنا - لسنوات عديدة - أن ندخل في حسابنا المعدّل الثابت في ازدياد سرعات ساعة المعالج. غير أن الحدود الفيزيائية مثل عائلنا أساسياً في طريق التزايد المستمر في سرعات الساعة؛ وذلك بسبب تزايد كثافة الطاقة بمعدل لاخطي يفوق تزايد سرعة الساعة، حيث تتعرض الرقاقات chips لخطر الانصهار بمجرد أن تصبح سرعات ساعتها كبيرة كفاية. ولتنجيز حسابات أكثر بالثانية، لا تُصنّف الرقاقات لتحتوي نواة معالجة واحدة فقط بل عدة نوى معالجة. ونستطيع أن نشبه هذه الحواسيب المتعددة النوى multicore بعدة حواسيب تسلسلية على رقاقة مفردة؛ وبكلمات أخرى، هي نوع من الحواسيب المتوازية parallel computers. وللحصول على التنجيز الأفضل للحواسيب المتعددة النوى، نحتاج عادةً إلى تصميم خوارزميات ونحن نضع فكرة التوازي في أذهاننا. يعرض الفصل 27 نموذجاً للخوارزميات "المتعددة التياصب multithreaded"، التي تستفيد من النوى المتعددة. لهذا النموذج فوائد من وجهة نظر نظرية، ويشكل القاعدة لعدة برامج حاسوب ناجحة، تشمل برنامجاً ليطوبة الشطرنج.

## تمارين

### 1-1.1

أعط مثلاً واقعياً يحتاج إلى الفرز، أو مثلاً واقعياً يحتاج إلى حساب الغلاف المحدب  $convex hull$ .

### 2-1.1

ما هي قياسات الفعالية الأخرى غير السرعة التي قد نستخدم في الواقع؟

### 3-1.1

اخر بنية معطيات عاينتها سابقاً، وناقش نقاط قوتها وحدود استخدامها.

### 4-1.1

كيف تكون مألناً أقصر مسار والبالغ الجوال الوردتان آنفاً متشابهتين؟ وكيف تكونان مختلفتين؟

### 5-1.1

ابحث في مسألة حقيقية لا بد من البحث عن حلها الأفضل، ثم ابحث في مسألة يكون أفضل حل تقريبي لها حلاً جيداً إلى حد بعيد.

## الخوارزميات بصفته تقانة

2.1

إذا افترضنا أن سرعة الحواسيب لانهائية وأن ذاكرة الحواسيب مجانية، فهل ثمة سبب لدراسة الخوارزميات؟ الجواب نعم، لا لسبب إلا لأنك ترغب في إثبات أن طريقة الحل التي نقتربها تتوقف، وأنها تتوقف بإعطاء النتيجة الصحيحة.

إذا كانت سرعة الحواسيب لانهائية، فإن أية طريقة صحيحة لحل مسألة ما سوف تؤدي الغرض. لكن قد ترغب في أن يكون تنحيك للحل يطبق عملياً هندسة الريمجات (على سبيل المثال يجب أن يكون تنحيك مصمماً وموثقاً جيداً)، لكنك ستستخدم على الأغلب أسهل الطرق لتنحيها.

من البديهي أن الحواسيب قد تكون فائقة السرعة، لكنها ليست ذات سرعة لانهائية. وقد تكون ذاكرة الحاسوب غير غالية الثمن، لكنها ليست مجانية. لذلك فإن زمن الحساب هو مورد محدود، وكذلك حجم الذاكرة. ولهذا السبب ينبغي استخدام هذه الموارد بحكمة، وستساعدك الخوارزميات الفعالة في الزمن والذاكرة على تحقيق هذا الغرض.

## الفعالية

كثيراً ما تختلف الخوارزميات للخصصة لحل للمشكلة نفسها في فعاليتها  $efficiency$ . يمكن أن تكون هذه الاختلافات ملموسة أكثر من الاختلافات الناتجة عن البيئي للمادية  $hardware$  والبرمجيات  $software$ .



كمثال على ذلك، سنرى في الفصل الثاني خوارزميَّ فرز. تُعرَفُ الخوارزمية الأولى **بالفرز بالإدراج** *insertion sort*، ويستغرق تنفيذها زمنًا يعطى تقريبًا بالعلاقة  $c_1 n^2$ ، وذلك لترتيب  $\blacksquare$  عنصرًا، حيث  $c_1$  ثابت لا يعتمد على  $n$ . أي إذا تستغرق زمنًا يتناسب مع  $n^2$ . أما الخوارزمية الثانية، فهي **الفرز بالدمج** *merge sort*، ويستغرق تنفيذها زمنًا يعطى تقريبًا بالعلاقة  $c_2 n \lg n$ ، حيث  $\lg n$  تعني  $\log_2 n$ ، و  $c_2$  ثابت آخر لا يعتمد أيضًا على  $n$ . يكون ثابت الفرز بالإدراج عادةً أصغر من ثابت الفرز بالدمج، أي  $c_2 < c_1$ . وسنرى لاحقًا أن للعوامل الثابتة تأثيرًا على زمن التنفيذ أقل بكثير من تأثير حجم المسألة  $n$ . فإذا كتبنا زمن تنفيذ الفرز بالإدراج بالعلاقة  $c_1 n$ ، وزمن تنفيذ الفرز بالدمج بالعلاقة  $c_2 n \cdot \lg n$ ، عندها نجد أنه فيما يحتوي الفرز بالإدراج العامل  $n$  في زمن تنفيذه، يحتوي الفرز بالدمج العامل  $\lg n$  وهو أصغر بكثير من سابقه. (نلاحظُ إذا كان  $n = 1000$ ، فإن  $\lg n$  يساوي 10 تقريبًا، وإذا كان  $n = 10^6$ ، فإن  $\lg n$  لا يتجاوز 20.) إن الفرز بالإدراج ينقذ عادةً أسرع من الفرز بالدمج في حالات أحجام الدخول الصغيرة، ولكن ما إن يصبح حجم المسألة  $n$  كبيرًا كفاية، حتى تصبح فائدة العامل  $\lg n$  في الفرز بالدمج مقارنةً بـ  $n$  أكثر من تعويض الفرق في العوامل الثابتة  $c_1$  و  $c_2$ ؛ فهما كانت قيمة  $c_1$  أصغر من  $c_2$ ، فتوجد دومًا نقطة تقاطع يكون بعدها الفرز بالدمج أسرع.

لنأخذ مثالًا واقعيًّا نقارن فيه بين حاسوب سريع نسبيًّا (حاسوب A) ينقذ خوارزمية الفرز بالإدراج، وبين حاسوب بطيء نسبيًّا (حاسوب B) ينقذ خوارزمية الفرز بالدمج. يُطلب إلى كلتا الخوارزميتين فرز صفيحة مكونة من عشرة ملايين عدد. (قد يبدو أن عشرة ملايين عدد هائل، لكن إذا كانت الأعداد صحيحة ومثلة باستخدام 8 بايت لكل منها، فيشغل الدخول نحوًا من 80 ميغا بايت، وهذا يمكن تخزينه حتى في ذاكرة الحاسوب المحمول الرخيص الذي يتسع لأضعاف هذا الحجم.) لنفترض أن الحاسوب A ينقذ 10 مليارات تعليمة في الثانية (أسرع من أي حاسوب تسلسلي بمفرده في زمن تأليف هذا الكتاب)، والحاسوب B ينقذ فقط عشرة ملايين (10<sup>7</sup>) تعليمة في الثانية، أي إن الحاسوب A أسرع من الحاسوب B بألف مرة من حيث القدرات الحاسوبية الأولية. ولجعل الفرق أكثر وضوحًا، لنفترض أن أشهر مبرمج في العالم يَنتِج خوارزمية الفرز بالإدراج بلغة الآلة الخاصة بالحاسوب A، وأن الرماز الناتج يتطلب  $2n^2$  تعليمة لفرز  $n$  عددًا (هنا  $c_1 = 2$ )، ولنفترض إضافةً إلى ذلك أنه يَنتِج خوارزمية الفرز بالدمج مبرمج متوسط المهارة بلغة برمجةٍ عليها تُستخدم مبرمج غير فعال بحيث يُنتج رمزًا يتطلب  $50n \lg n$  تعليمة. عندها نجد أنه لترتيب 10 ملايين عدد يستغرق الحاسوب A:

$$\frac{\text{تعلّمة } 2 \cdot (10^7)^2}{\text{تعلّمة/ثانية } 10^{10}} = \text{, (أكثر من 5.5 ساعة) ثانية } 20,000$$

ويستغرق الحاسوب B:

$$\frac{\text{تعليلة } 50 \cdot 10^7 \lg 10^7}{\text{تعليلة/ثانية } 10^7} \approx \text{أقل من 20 دقيقة} \approx 1163 \text{ ثانية}$$

وهكذا نجد أنه باستخدام خوارزمية ذات زمن تنفيذ بطيء، ومترجم ضعيف، فإن الحاسوب B ينقذ هذه الخوارزمية أسرع بـ 17 مرة من الحاسوب A! على أن فائدة الفرز بالدمج تظهر بوضوح أكبر في حال فرز 100 مليون عدد: ففي حين يستغرق الفرز بالإدراج أكثر من 23 يومًا، يستغرق الفرز بالدمج أقل من أربع ساعات. وبوجه عام، تزداد الفائدة النسبية للفرز بالدمج بازدياد حجم المسألة.

### الخوارزميات والتقانات الأخرى

يُظهر المثال السابق أنه ينبغي أن نتعامل مع الخوارزميات على أنها *تقانة technology* شأنها في ذلك شأن الكيان المادي للحاسوب. يعتمد الأداء العام للنظام على اختيار خوارزميات فعالة بقدر اعتماده على اختيار الكيان المادي السريع للحاسوب. وقد حدث تقدم سريع في الخوارزميات تمامًا كما حدث في تقانات الحاسوب الأخرى.

قد تتساءل: هل الخوارزميات هامة بالفعل في الحواسيب المعاصرة مع وجود التقانات المتقدمة الأخرى مثل:

- بنى الحاسوب المتقدمة وتقانات التصنيع.
- واجهات المستخدم البيانية المألوفة وسهولة الاستخدام GUI.
- النظم الغرضية التوجه Object Oriented Systems.
- تقانات الويب المكاملة Integrated Web Technologies.
- التشبيك السريع، السلكي واللاسلكي Fast Networking, both Wired and Wireless.

الجواب نعم. لأنه على الرغم من أن وجود بعض التطبيقات التي لا تتطلب صراحة محتوى خوارزميًا في مستوى التطبيق (مثل بعض التطبيقات البسيطة المعتمدة على الويب)، فإن كثيرًا منها يتطلب ذلك. لنأخذ على سبيل المثال خدمة معتمدة على الويب تحدد كيفية السفر من مكان إلى آخر. إن تحقيق هذه الخدمة قد يقوم على كيان مادي سريع، وواجهة مستخدم بيانية، وتشبيك واسع، وكذلك على إمكانية التوجه بالأغراض. إلا أنه يتطلب أيضًا خوارزميات لبعض العمليات، مثل إيجاد الطرق (ربما استخدام خوارزمية حساب أقصر مسار)، وإظهار خرائط الترجمة، واستقراء العناوين.

أضف إلى ذلك أن التطبيق، ولو كان لا يتطلب محتوى خوارزميًا في مستوى التطبيق، فإنه يعتمد بقوة

على الخوارزميات. فالتطبيق يُعتمد على كيانٍ ماديٍّ سريع، وتصميمُ الكيانِ المادي يُستخدم بدوره الخوارزميات. والتطبيق يُعتمد على واجهات مستخدمٍ بيانية، وتصميمُ أية واجهةٍ بيانية يُعتمد بدوره على الخوارزميات. والتطبيق يُعتمد على الشبكات، والتسيير routing يُعتمد بدوره بكثافة على الخوارزميات. وأخيراً، التطبيق يُكتب بلغة رماز الآلة machine code، أي يعالجه مترجمٌ compiler أو مفسرٌ interpreter أو مُجمّع assembler، وجميعها تُعتمد بدورها بكثافة على الخوارزميات. وهكذا فإن الخوارزميات موجودة في صلب معظم التقانات المستخدمة في الحواسيب المعاصرة.

يضاف إلى ذلك أنه مع الازدياد المستمر في قدرات الحواسيب، فإننا نستخدمها لحل مسائل أكبر لم يسبق لنا أن حلناها. وكما رأينا في المقارنة بين الفرز بالإدراج والفرز بالدمج، تصبح الفروق في الفعالية بين الخوارزميات محسوسة بوضوح مع أحجام للسائل الكبيرة.

إن امتلاك قاعدة متينة من المعرفة والتقنية الخوارزمية تقلل عاملَ فضلٍ لتمييز المبرمجين المهرة حقاً من المبتدئين؛ فباستخدام ثقافة حوسبة حديثة يمكنك إنجاز بعض المهامات دون معرفة الكثير عن الخوارزميات، ولكنك تستطيع إنجاز الكثير الكثير إذا امتلكت خلفية خوارزمية جيدة.

## تعاريف

### 1-2.1

أعط مثالاً عن تطبيق يتطلب معنًى خوارزميةً في مستوى التطبيق، وناقش دور الخوارزميات المعنية.

### 2-2.1

افترض أننا نقارن بين تميز الفرز بالإدراج والفرز بالدمج على الآلة نفسها. يتم الفرز بالإدراج بـ  $8n^2$  خطوةً لمدخلات حجمها  $n$ ، على حين يتم الفرز بالدمج بـ  $64n \lg n$  خطوة. ما هي قيم  $n$  التي يتفوق فيها الفرز بالإدراج على الفرز بالدمج؟

### 3-2.1

ما هي أصغر قيمة لـ  $n$  بحيث يكون تنفيذ خوارزمية زمن تنفيذها  $100n^2$  أسرع من خوارزمية زمن تنفيذها  $2^n$  على الحاسوب نفسه؟

## مسائل

### 1-1 مقارنة زمن التنفيذ

حدّد - لكل دالة  $f(n)$  وزمن  $t$  في الجدول التالي - أكبر قيمة لـ  $n$  لمسألة يمكن حلها خلال الزمن  $t$ ، بافتراض أن الخوارزمية المستخدمة لحل المسألة تستغرق  $f(n)$  ميكروثانية.

1	1	1	1	1	1	1	
قرن	سنة	شهر	يوم	ساعة	دقيقة	ثانية	
							$\lg n$
							$\sqrt{n}$
							$n$
							$n \lg n$
							$n^2$
							$n^3$
							$2^n$
							$n!$

## ملاحظات الفصل

هناك الكثير من مصادر المعلومات للمنازة عن موضوع الخوارزميات العام، منها: Aho و Hopcroft و Ullman [5, 6]؛ Baase و Van Gelder [28]؛ Dasgupta و Brassard و Bratley [55]؛ Dasgupta و Papdimitriou و Vazirani [83]؛ Goodrich و Tamassia [148]؛ Hofri [178]؛ Horowitz و Sahni و Rajasekaran [181]؛ Johnsonbaugh و Schaefer [193]؛ Kingston [205]؛ Kleinberg و Tardos [208]؛ Knuth [209, 210, 211]؛ Kozen [220]؛ Levitin [235]؛ Manber [242]؛ Mehlhorn [249, 250, 251]؛ Purdom و Brown [287]؛ Reingold و Nievergelt و Deo [293]؛ Sedgewick [306]؛ Flajolet و Skiena [307]؛ Wilf [356]. وناقش كلٌّ من Bentley [42, 43] و Gonnet [145] بعضُ أكثر الجوانب عمليةً في تصميم الخوارزميات. توجد أيضاً دراسات تسمح بحل الخوارزميات في كتيب علم الحاسوب النظري *Handbook of Theoretical Computer Algorithms and Theory of* CRC في الخوارزميات ونظرية الحوسبة [342] *Science, Volume A*، وكتيب CRC في الخوارزميات ونظرية الحوسبة [342] *Computation Handbook* [25]. وتوجد أيضاً نسخة عن الخوارزميات المستخدمة في علم الأحياء المحسوب في كتيب Gusfield [156] و Pevzner [275] و Setubal and Meidanis [310] و Waterman [350].

سيطلبك هذا الفصل على المنهجية التي سنستخدمها في هذا الكتاب للتفكير في تصميم وتحليل الخوارزميات. هذا الفصل مستقل بذاته، لكنه يحتوي عدة إشارات لمقاطع ستعرض في الفصولين الثالث والرابع. (يحتوي كذلك عدة مجاميع سلمية، يبين للمحق (أ) كيفية حلها.)

سنبداً بتفحص خوارزمية الفرز بالإدراج insertion sort لحل مسألة الفرز المعروضة في الفصل الأول. نعرف "شبه الرماز" pseudocode الذي ينبغي أن يكون معروفاً للقراء الذين عملوا في برمجة الحاسوب، ونستخدمه لإيضاح كيف سنوصف خوارزمياتنا. بعد توصيف الخوارزمية، نبرهن صحة فرزها، ونحلل زمن تنفيذها. يقدم التحليل تدويناً notation يسلط الضوء على كيفية ازدياد زمن التنفيذ مع زيادة عدد الحدود التي يجب فرزها. بعد مناقشة الفرز بالإدراج، نعرض مفهوم ترقُّق تُشَدُّ لتصميم الخوارزميات ونستخدمه لتطوير خوارزمية تسمى الفرز بالدمج merge sort. ثم نتهي الفصل بتحليل زمن تنفيذ خوارزمية الفرز بالدمج.

## 1.2 الفرز بالإدراج

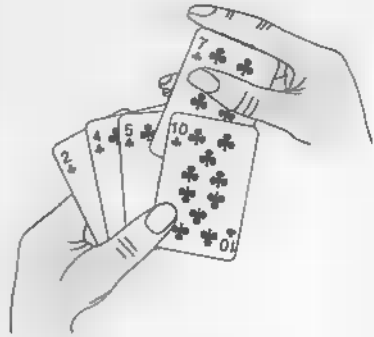
نحل خوارزمتنا الأولى، الفرز بالإدراج، مسألة الفرز sorting problem الواردة في الفصل الأول:

الدخل: متالية من  $n$  عدداً  $(a_1, a_2, \dots, a_n)$ .

المخرج: تبديل (إعادة ترتيب)  $(a'_1, a'_2, \dots, a'_n)$  متالية الدخل بحيث يكون  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .

نُعرفُ الأعداد التي نرغب في ترتيبها أيضاً بالمفاتيح Keys. ومع أننا من حيث المبدأ نرتب متالية، غير أن الدخل الذي يرد إلينا يكون على شكل صفيحة من  $n$  عنصرًا.

سنصف في هذا الكتاب الخوارزميات عموماً كبرامج مكتوبة بشبه رماز pseudocode يشابه في عدة جوانب C، أو C++، أو Java، أو Python، أو Pascal. فإذا كنت قد اطلعت على أيٍّ من هذه اللغات، فلن تجد عناء كبيراً في قراءة خوارزمياتنا. إن ما يفرِّق شبه الرماز عن الرماز "الحقيقي"، هو أننا نستخدم في شبه الرماز الطريقة المعيرة الأكثر وضوحاً وإيجازاً لتوصيف الخوارزمية المخططة. قد تكون اللغة الإنكليزية هي



الشكل 1.2 ترتيب أوراق اللعب باستخدام الفرز بالإدراج

أوضح طريقة، لذا لا تندهش إذا صادفت عبارة أو جملة إنكليزية مضمنة في مقطع من الرمز "الحقيقي". ثمة فرق آخر بين شبه الرمز والرمز الحقيقي، هو أن شبه الرمز لا يُعنى عادةً بقضايا هندسة البرمجيات. وللتعبير عن جوهر الخوارزمية بإيجاز أكبر، غالباً ما تتجاهل قضايا تجريد المعطيات، والنسبية modularity، ومعالجة الخطأ.

سنبدأ بالفرز بالإدراج insertion sort، الذي هو خوارزمية فعالة لفرز عدد صغير من العناصر. يعمل الفرز بالإدراج وفق الطريقة التي يفرز بها كثير من لاعبي الورق أوراق اللعب. نبدأ بيد يسرى فارغة وأوراق اللعب وجهها إلى الأسفل باتجاه الطاولة. ثم نرفع في كل مرة ورقة واحدة من الطاولة ونُدرجها (نُغشِرها) في الموضع الصحيح في اليد اليسرى. لإيجاد الموضع الصحيح للورقة، نقارنهما بكل ورقة في اليد، من اليمين إلى اليسار، كما هو موضح في الشكل 1.2. وتكون الأوراق الموجودة في اليد اليسرى مغروزة في كل المرات، وكانت هذه الأوراق أصلاً الأوراق العلوية للكومة الموجودة على الطاولة.

سنعرض شبه رمازنا لخوارزمية الفرز بالإدراج على هيئة إجراء اسمه INSERTION-SORT، يأخذ متوسطاً هو صيغة  $A[1..n]$  تحتوي على متالية طولها  $n$  يُطلب ترتيبها. (يشار في الرمز إلى عدد العناصر  $n$  في  $A$  بـ  $A.length$ ). تُفرز الخوارزمية أعداداً الدخول في المكان in place: وتعيد ترتيب الأعداد ضمن الصيغة  $A$ ، مع تخزين عدد ثابت منها على الأكثر خارج الصيغة في أي وقت. عندما ينتهي الإجراء INSERTION-SORT سوف تحتوي صيغة الدخول  $A$  متالية الخرج للفرز.

INSERTION-SORT( $A$ )

1 for  $j = 2$  to  $A.length$

2      $key = A[j]$

3     // Insert  $A[j]$  into the sorted sequence  $A[1..j-1]$ .

```

4      i = j - 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i - 1
8      A[i + 1] = key

```

### لامتغيرات الحلقة وصحة الفرز بالإدراج

يوضح الشكل 2.2 كيف تعمل هذه الخوارزمية في حالة الصيغة  $A = (5, 2, 4, 6, 1, 3)$ . يشير الدليل  $j$  إلى "الورقة الحالية" التي تدرج في اليد. في بداية كل تكرار من تكرارات حلقة **for** التي متحول تحكمها هو  $j$ ، تكون الصيغة الجزئية المؤلفة من العناصر  $A[1..j-1]$  أوراق اليد المفروزة حالياً، وتطابق الصيغة الجزئية المتبقية  $A[j+1..n]$  كومة الأوراق التي مازال على الطاولة. إن العناصر  $A[1..j-1]$  هي في الواقع نفس العناصر التي في المواضع  $1$  حتى  $j-1$  أصلاً، لكنها الآن بترتيب مفروز. نصوص هذه الخواص للصيغة الجزئية  $A[1..j-1]$  صورياً بصفتها **لامتغير حلقة invariant loop**:

في بداية كل تكرار من تكرارات حلقة **for** التي تشمل الأسطر 1-8، تتألف الصيغة الجزئية  $A[1..j-1]$  من العناصر التي هي أصلاً في هذه الصيغة الجزئية، لكن بترتيب مفروز.

نستخدم لامتغيرات الحلقة لتساعدنا على فهم سبب كون خوارزمية ما صحيحة. وهنا يجب أن نوضح ثلاثة أمور عن لامتغير الحلقة:

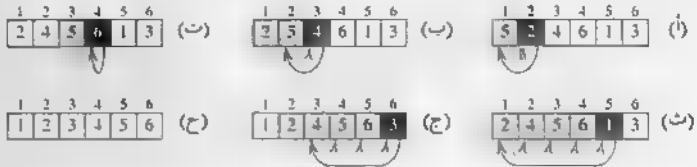
الاستعداد: يكون لامتغير الحلقة صحيحاً قبل التكرار الأول للحلقة.

المحافظة: إذا كان لامتغير الحلقة صحيحاً قبل تكرار ما للحلقة، فيبقى صحيحاً قبل التكرار التالي.

الإنهاء: عندما تنتهي الحلقة، يغطي لامتغير الحلقة خاصية مفيدة تساعد على بيان أن الخوارزمية صحيحة.

عند تحقق الخاصيتين الأولى والثانية، يكون لامتغير الحلقة صحيحاً قبل كل تكرار من تكرارات الحلقة. (طبعاً، لدينا الحرية في استخدام خصائص مختلفة أخرى غير لامتغير الحلقة نفسه لإثبات أن لامتغير الحلقة يبقى صحيحاً قبل كل تكرار.) لاحظ الشبه مع الاستقراء الرياضي، وهو أنه كي نثبت تحقق خاصية ما، ينبغي أن نثبت حالة أساسية وخطوة استقرائية. وهنا يظهر أن اللامتغير يتحقق قبل أن يقابل التكرار الأول للحالة الأساسية، ويظهر أيضاً أن اللامتغير يتحقق من تكرار إلى تكرار يقابل الخطوة الاستقرائية.

الخاصية الثالثة ربما تكون أهم خاصية، لأننا نستخدم لامتغير الحلقة لبيان الصحة. نستخدم عادة لامتغير الحلقة بالانحياز مع الشرط للسبب لإنهاء الحلقة. تختلف خاصية الإنهاء عن الكيفية التي نستخدم فيها عادة الاستقراء الرياضي، حيث نطبق الخطوة الاستقرائية حتى اللانهاية؛ أما هنا، فنوقف "الاستقراء" عندما تنتهي الحلقة.



**الشكل 2.2** عمل الفرز بالإدراج على الصفقة  $A = \{5, 2, 4, 6, 1, 3\}$ . تُظهر دلائل العناصر فوق المستطيلات، وتُظهر القيم المخزنة في مواضع الصفقة داخل المستطيلات. تمثل الأشكال من (أ) إلى (ج) تكرارات حلقة `for` التي تشمل الأسطر 8-1. يحتفظ المستطيل الأسود في كل تكرار بالفتاح (key) للأخوذ من  $A[j]$ ، الذي يُقارن بالقيم الموجودة في المستطيلات المظلمة إلى يساره في اختبار السطر 5. تُظهر الأسهم المظلمة قيم الصفقة المزاخمة موضعًا واحدًا إلى اليمين في السطر 6، وتُشير الأسهم السوداء إلى أين يتحرك الفتاح في السطر 8. يمثل الشكل (ج) الصفقة النهائية المفروزة.

لنفحص كيفية تحقيق هذه الخواص في حالة الفرز بالإدراج.

**الاستدعاء:** نبدأ ببيان أن لامتغير الحلقة يتحقق قبل بدء التكرار الأول للحلقة، عندما يكون  $j = 2$ <sup>1</sup>. لذلك تتألف الصفقة الجزئية  $A[2..j-1]$  من العنصر  $A[1]$  فقط، الذي هو في الواقع العنصر الأصلي في  $A[1]$ . إضافة إلى ذلك، فإن هذه الصفقة الجزئية مفروزة (بداية، بالطبع)، وهذا يبيّن أن لامتغير الحلقة يتحقق قبل بدء التكرار الأول للحلقة.

**المحافظة:** بعد ذلك، نتناول الخاصية الثانية: أي بيان أن كل تكرار يحافظ على لامتغير الحلقة. بعبارة بسيطة، يقوم جسم الحلقة `for` على إزاحة  $A[j-1]$ ، و  $A[j-2]$ ، و  $A[j-3]$  موقعًا واحدًا نحو اليمين المرة تلو الأخرى حتى يُجد الموقع المناسب للعنصر  $A[j]$  (الأسطر 4-7)، وعندما يُدرج قيمة  $A[j]$  في هذا الموقع (السطر 8). تتألف عندها الصفقة الجزئية  $A[1..j]$  من العناصر الموجودة أصلاً في  $A[1..j-1]$ ، لكن بترتيب مفروز. ويحافظ تزايد  $j$  في حالة التكرار التالي للحلقة `for` على لامتغير الحلقة.

تتطلب منا معالجة أكثر صعوبة للخاصية الثانية تحديد وبيان لامتغير حلقة حلقة `while` الواقعة في الأسطر 5-7. عند هذه النقطة نقضل عدم الفوص حتى هذه الدرجة من الصورة، ونكتفي بتحليلنا المبسط لبيان أن الحلقة الخارجية تحقق الخاصية الثانية.

**الإنهاء:** أخيرًا لنفحص ما يُحصل عندما تنتهي الحلقة. إن الشرط الذي يسبب انتهاء حلقة `for` هو

<sup>1</sup> عندما تكون الحلقة هي حلقة `for`، فإن اللحظة التي نختبر عندها لامتغير الحلقة - وهي بالضبط اللحظة التي تسبق أول تكرار - هي مباشرة بعد الإسناد الابتدائي لتحول عداد الحلقة، وبالضبط قبل أول اختبار في ترويسة الحلقة. في حالة الفرز بالإدراج، يكون هذا الزمن بعد إسناد 2 للمتحول  $j$  لكن قبل أول اختبار فيما إذا كان  $j \leq A.length$ .



$n = A.length > z$ . ولما كان كل تكرار في الحلقة يزيد قيمة  $z$  بمقدار 1، فيجب أن نحصل على  $z = n + 1$  في ذلك الحين. وبتعويض القيمة  $n + 1$  عوضاً عن  $z$  في عبارة لامتغير الحلقة، نحصل على الصيغة الجزئية  $A[1..n]$  وهي تتألف من العناصر الموجودة في  $A[1..n]$  أصلاً، ولكن بترتيب مفرود. وملاحظة أن الصيغة الجزئية  $A[1..n]$  هي الصيغة الداخلية، نستنتج أن الصيغة الداخلية مفرودة. فالخوارزمية إذن صحيحة.

نستخدم طريقة لامتغيرات الحلقة هذه لإظهار الصحة في هذا الفصل، وفي الفصول القادمة أيضاً.

### اصطلاحات شبه الرماز

سوف نستخدم الاصطلاحات التالية في شبه رمازنا.

- تشير الإزاحة العمودية هنا، الكتابة إلى بنية كتلة. فمثلاً، يتألف جسم حلقة **for** الذي يبدأ عند السطر 1 من الأسطر 2-8، أما جسم حلقة **while** التي تبدأ عند السطر 5، فيحتوي السطرين 6 و 7، ولكن لا يحتوي السطر 8. ينطبق غرضنا في الإزاحة العمودية أثناء الكتابة على تعليمات **if-else**<sup>2</sup> أيضاً. إن استخدام الإزاحة العمودية عوضاً عن المؤشرات الاصطلاحية لبنية الكتلة، مثل تعليمتي **begin** و **end**، يقلل كثيراً من مراكمة المصطلحات الإضافية على حين أنه يحافظ على الوضوح، أو حتى يزيد منه.<sup>3</sup>
- تمتلك التراكيب الحلقية **while**، و **for**، و **repeat-until**، وتركيب الشرط **if-else** معاني مشابهة لمعانيها في لغات البرمجة C، و C++، وجافا، و python، وباسكال.<sup>4</sup> في هذا الكتاب، يحتفظ عداد الحلقة بقيمته بعد الخروج من الحلقة، على عكس بعض الحالات التي تظهر في C++ وجافا وباسكال. وهكذا، تكون قيمة عداد الحلقة، مباشرة بعد حلقة **for**، هي أول قيمة تتجاوز حد حلقة **for**. لقد استخدمنا هذه الخاصية في برهاننا صحة الفرز بالإدراج. إن ترويسة حلقة **for** في السطر 1 هي  $for\ j = 2\ to\ A.length$ ، وهذا يعني أنه عندما تنتهي هذه الحلقة، يكون  $j = A.length + 1$  (أو،

<sup>2</sup> تعليمية **if-else**، نضع **else** عمودياً إلى نفس مستوى **if** المطابقة لها. وعلى الرغم من أننا نحذف الكلمة المفتاحية **then**، فإننا نشير أحياناً إلى الجزء للنفذ - عندما تكون نتيجة الاختيار الذي يتبع **if** صحيحة - على أنه عبارة **then**. في حالة الاختبارات المتعددة الفروع، نستخدم **elseif** لنشير إلى الاختبارات الواردة بعد الاختيار الأول.

<sup>3</sup> يظهر كل إجراء شبه رمازي في هذا الكتاب على صفحة واحدة بحيث لا تضطر للتمييز بين مستويات الإزاحة العمودية في رماز مفرق على صفحات.

<sup>4</sup> تمتلك معظم اللغات للفيكيلة ككل **block-structured languages** تراكيب مكافئة، لذا قد تختلف القواعد الدقيقة لصياغتها. فننقد لغة python حلقات **repeat-until**، وتختلف حلقات **for** قليلاً في طريقة عملها عن حلقات **for** في هذا الكتاب.

ما يكافئ  $j = n + 1$  ، لأن  $n = A.length$  ). نستخدم الكلمة المفتاحية **to** عندما نريد حلقة **for** عدداً حلقتها في كل تكرار، ونستخدم الكلمة المفتاحية **downto** عندما تُنقص حلقة **for** عدداً حلقتها. وعندما يتغير عددا الحلقة بمقدار أكبر من 1، يُستخدَم مقدار التغير الكلمة المفتاحية الاختيارية **by**.

- يشير الرمز **"/"** إلى أن ما تبقى من السطر هو تعليق.
- الإسناد المتعدد من الشكل  $i = j = e$  يستد لُكل من المتحولين  $i$  و  $j$  قيمة العبارة  $e$ ؛ ويجب أن تعالج كإسناد  $e = f$  متبوع بالإسناد  $i = j$ .
- المتحولات (مثل  $i$  و  $j$  و **key**) هي متحولات عملية بالنسبة إلى إجراء معطى. ولن نستخدم متحولات عامة (global variables) دون الإشارة إليها صراحة.
- يجري الوصول إلى عناصر الصيغة بتحديد اسم الصيغة متبوعاً بالدليل ضمن قوسين مربعين. مثال،  $A[i]$  يشير إلى العنصر ذي الترتيب  $i$  من الصيغة  $A$ . الكتابة بالشكل  $A[i]$  تستخدم للتعبير عن مجال من القيم ضمن صيغة. ويُستخدم التدوين  $A[i]$  للإشارة إلى مجال من القيم ضمن صيغة؛ وعلى ذلك تشير  $A[1..j]$  إلى صيغة جزئية من  $A$  تتألف من  $j$  عنصراً هي:  $A[1], A[2], \dots, A[j]$ .
- نظم المعطيات المركبة عادةً ضمن **أغراض** **Objects**، تتكون بدورها من **واصفات** **attributes**. نُنفذُ إلى واصف محدد باستخدام التركيب النحوي الموجود في كثير من لغات البرمجة الغرضية التوجه: اسم الغرض، متبوعاً بنقطة، متبوعاً باسم الواصف. كمثل على ذلك، نعالج الصيغة كغرض مع صفة الطول **length** للإشارة إلى عدد العناصر التي تحتويها. لتحديد عدد العناصر في صيغة  $A$  نكتب  $A.length$ .
- وتُعالج المتحول الذي يمثل صيغة أو غرضاً كمؤشر للمعطيات المثلة للصيغة أو الغرض. إن الإسناد  $y = x$ ، يؤدي إلى جعل  $y.f$  تساوي  $x.f$  لكل الواصفات  $f$  لغرض ما  $x$ . إضافة إلى ذلك، إذا جعلنا الآن  $x = 3$ ،  $x.f = 3$ ، بعدئذٍ لن يكون  $x.f = 3$  فقط، بل  $y.f = 3$  أيضاً. وبعبارة أخرى، تشير كل من  $x$  و  $y$  إلى الغرض نفسه بعد الإسناد  $y = x$ .
- يمكن أن يتتالي تدويننا للواصفات "cascade". فعلى سبيل المثال افترض أن الواصف  $f$  نفسه هو مؤشر لنوع ما من الأغراض  $g$ . عندها يحتوي التدوين  $x.f.g$  ضمناً أقواساً من الشكل  $(x.f).g$ . وبعبارة أخرى، إذا أسندنا  $y = x.f$ ، عندها تكون  $x.f.g$  هي  $y.g$  نفسها.
- قد لا يشير المؤشر أحياناً إلى أي غرض على الإطلاق. في هذه الحالة، نعطي القيمة الخاصة NIL.
- تُمرَّرُ المتوسطات إلى الإجراء بالقيمة **by value**: يتلقى الإجراء المُستدعى نسخة الخاصة من المتوسطات، وإذا أسند قيمةً لموسط، فلن يَرَى الإجراء المُستدعى التغير الحاصل. وعندما تُمرَّرُ أغراض، يُستخدَم المؤشر للمعطيات المُختلفة للغرض، لكن لا تنسخ واصفات الغرض. فعلى سبيل المثال، إذا كان  $x$  موسطاً

لإجراء مُستدعى، فلن يكون الإسناد  $x = y$  داخل الإجراء المُستدعى مرتباً للإجراء المُستدعى. أما الإسناد  $x, f = 3$ ، فهو مرثي. وبالمثل، تُعزّز الصفائف بالموشر، بحيث يُؤزّر مؤشر إلى صفيقة، بدلاً من الصفيقة كلها، وتكون تغيرات عناصر الصفيقة الفردية مرتبة للإجراء المستدعي.

- تعيد تعليمة **return** التحكم إلى نقطة الاستدعاء في الإجراء المُستدعي. تأخذ أيضاً معظم تعليمات **return** قيمةً لتعيدها إلى المستدعي. يختلف شبه رمازنا عن كثير من لغات البرمجة في أننا نسمح بأن تعاد قيم متعددة في تعليمة **return** مفردة.
- تصنف للعاملات البولائية "and" و "or" مثل **دارات قصر short circuiting**. وهذا يعني أنه عندما نقيم العبارة "x and y" فإننا نقيم x أولاً. فإذا كانت قيمة x هي FALSE، فلا يمكن عندها تقييم كامل العبارة بـ TRUE، لذا لا نقيم y. من ناحية أخرى، إذا كانت قيمة x هي TRUE، فعلىنا تقييم y لتحديد قيمة العبارة كاملة. وبالمثل، نقيم y في العبارة "x or y" إذا كانت قيمة x هي FALSE فقط. نسمح لنا عوامل دارات القصر بكتابة عبارات بولائية مثل: " $x \neq \text{NIL and } x.f = y$ " دون القلق مما يمكن أن يحدث عندما نحاول تقييم  $x.f$  عندما تكون x هي NIL.
- تشير الكلمة المفتاحية **error** إلى أن خطأ ما قد حدث لأن شروط استدعاء الإجراء كانت خاطئة. ويكون الإجراء المستدعي هو المسؤول عن معالجة الخطأ، ومن ثم فإننا لا نحدد الفعل الذي يجب أن يتخذ.

## تمارين

### 1-1.2

باستخدام نموذج الشكل 2.2، وضّح عمل الفرز بالإدراج على الصفيقة  $A = \{31, 41, 59, 26, 41, 58\}$ .

### 2-1.2

أعد كتابة إجراء الفرز بالإدراج للفرز بترتيب متناقص بدل الفرز بترتيب متزايد.

### 3-1.2

لتكن لدينا مسألة البحث **searching problem** التالية:

**الدخل:** متتالية من  $n$  عدداً  $\{a_1, a_2, a_3, \dots, a_n\}$  وقيمة ما  $v$ .

**الخروج:** دليل ما  $i$  بحيث  $v = A[i]$ ، أو القيمة الخاصة NIL في حال عدم ظهور  $v$  في  $A$ .

اكتب شبه رماز يحل البحث الخطي **linear search**، الذي يمسخ للمتتالية  $A$ ، بحثاً عن  $v$ . برهن باستخدام لامتنعير الحلقة أن خوارزمتك صحيحة. تأكد أن لامتنعير حلقتك يحقق الخصائص الثلاث الضرورية.

## 4-1.2

لتكن لدينا مسألة جمع عددين صحيحين ثنائيين كل منهما مكون من  $n$  بتًا، نُحَرِّثُ في صيفيتين  $A$  و  $B$ ، تتألف كلٌّ منهما من  $\blacksquare$  عنصرًا. يجب أن نُحَرِّثَ مجموع العددين الصحيحين بصيغة ثنائية في صيغة  $C$  عدد عناصرها  $(n + 1)$ . صُغِ المسألة صوريًا، واكتب شبه رماز لجمع العددين الصحيحين.

## 2.2 تحليل الخوارزميات

أصبح تحليل *analyzing* الخوارزمية يعني التنبؤ بالموارد التي تحتاج إليها الخوارزمية عند تنفيذها. في بعض الأحيان، تشكل الموارد مثل الذاكرة، أو عرض حزمة الاتصال، أو البنى المادية للحاسوب للاهتمام الرئيس، لكن في معظم الأحيان يكون زمن الحساب هو الذي نريد قياسه. يمكننا عمومًا - عن طريق تحليل عدة خوارزميات مرشحة لحل مسألة ما - أن نُحَدِّدَ بسهولة أيّ الخوارزميات أكثر كفاءةً. وقد يُظهر هذا التحليل أكثر من حلٍّ قابلٍ للتطبيق، لكنّ يمكننا أيضًا في كثير من الأحيان أن ننبه عدة خوارزميات أقل كفاءةً.

قبل أن نستطيع تحليل خوارزمية ما، يجب أن نمتلك نموذجًا لتقانة التنفيذ *implementation* التي سنستخدمها، ومن ذلك نموذج موارد هذه التقانة وكلفتها. سنفترض في معظم هذا الكتاب، معالجةً وحيثًا عموميًا، ونموذج آلة وصول عشوائي *random-access machine (RAM)* للمخوّصة باعتبارها تقانة تنّجيز، مدركين أن خوارزمياتنا ستُنفّذ باعتبارها برامج حاسوبية. تنفذ التعليمات في نموذج RAM تعليمة بعد تعليمة، دون وجود عمليات متساية *concurrent operations*.

إذا أردنا الحديث بدقة مطلقة، فعلى تعريف تعليمات نموذج RAM وكلفتها بدقة. إلا أن ذلك قد يكون مرهقًا، ولن يقدم إلا القليل من الرؤية داخل تحليل الخوارزمية وتصميمها. لذا يجب أن نكون متنبهين لعدم إساءة استعمال نموذج RAM. مثلاً، ماذا لو احتوت RAM تعليمة تقوم بالفرز؟ بمقدورنا عندئذ أن ننقذ عملية الفرز بتعليمة واحدة فقط. لن تكون مثل هذه الآلة واقعية، لأن الحواسيب الحقيقية لا تمتلك مثل هذه التعليمات. مرشدنا إذن في اختيار النموذج هو تصميم الحواسيب الحقيقية. يحتوي نموذج RAM تعليمات توجد عادة في الحواسيب الحقيقية: تعليمات حسابية (مثل: *add* و *subtract* و *multiply* و *divide* و *remainder* و *floor* و *ceiling*)، وتعليمات نقل معطيات (*load* و *store* و *copy*)، وتعليمات تحكم (التفرع الشرطي وغير الشرطي *conditional and unconditional branch*، واستدعاء الإجراء الفرعي *subroutine call*، والعودة *return*). تستغرق كلّ تعليمة منها مقدارًا ثابتًا من الزمن.

إن أنواع للمعطيات في نموذج RAM هي الأعداد الصحيحة *integer* وذات الفاصلة العائمة *floating point* (لتخزين الأعداد الحقيقية). ومع أننا في العادة لا نشغل أنفسنا بدقة تعريف الأعداد في هذا الكتاب، غير أن الدقة تكون في بعض التطبيقات حاسمة. نفترض أيضًا وجود حدٍّ لحجم كل كلمة من للمعطيات؛

فمثلاً، عند العمل على مدخلات من الحجم  $n$ ، نفترض عادة أن الأعداد الصحيحة تُخزن بالمقدار  $\lg n$  بتاً حيث  $c \geq 1$  ثابت. نحتاج إلى  $c \geq 1$  حتى نستطيع كل كلمة احتواء قيمة  $n$ ، متيحة لنا التدليل على عناصر الدخل المستقلة، ونشترط أن يكون  $c$  ثابتاً بحيث لا ينمو حجم الكلمة كقيماً. (لو أمكن لحجم الكلمة أن ينمو كقيماً، لاستطعنا تخزين كميات ضخمة من المعطيات في كلمة واحدة ولأمكننا أن نعمل عليها جميعها في زمن ثابت. ومن الواضح أن هذا السيناريو غير واقعي.)

تحتوي الحواسيب الحقيقية تعليمات لم تُدرج آنفاً، ومثل هذه التعليمات تُمثل المنطقة الرمادية في نموذج RAM. على سبيل المثال، هل الرفع إلى قوة تعليمية تستغرق زمناً ثابتاً؟ في الحالة العامة، لا؛ فهي تأخذ عدة تعليمات لحساب  $x^y$  عندما تكون  $x$  و  $y$  أعداداً حقيقية. من جهة ثانية، يكون الرفع إلى قوة، في بعض الحالات المحددة، عملية تستغرق زمناً ثابتاً. تمتلك كثير من الحواسيب تعليمية الإزاحة إلى اليسار "shift left"، التي تزيج في زمن ثابت بنات عدد صحيح  $k$  موضعاً إلى اليسار. في معظم الحواسيب، تكافئ عملية إزاحة بنات عدد صحيح موضعاً واحداً إلى اليسار الضرب بـ 2، أي إن إزاحة البنات  $k$  موضعاً إلى اليسار تكافئ الضرب بـ  $2^k$ . وهكذا، نستطيع مثل هذه الحواسيب حساب  $2^k$  بتعليمية زمن ثابت واحدة بإزاحة العدد الصحيح  $k$  موضعاً إلى اليسار، مادامت  $k$  ليست أكثر من عدد البنات في كلمة الحاسوب. سنحاول تجنب مثل هذه المناطق الرمادية في نموذج RAM، لكن سنعالج حساب  $2^k$  على أنها عملية زمن ثابت عندما يكون  $k$  عدداً صحيحاً موجباً وصغيراً كفاية.

لن نحاول في نموذج RAM نمذجة تراتبية الذاكرة memory hierarchy الشائعة في الحواسيب الحالية. ذلك أننا لا نمذج الذواكر الحامية caches memory أو الذاكرة الافتراضية virtual memory. نحاول بعض النماذج الحاسوبية أن تأخذ بالحسبان تأثيرات تراتبية الذاكرة، المأثرة أحياناً في البرامج الحقيقية على آلات حقيقية. جزء ضئيل فقط من مسائل هذا الكتاب يعالج آثار تراتبية الذاكرة، فيما لن يُعنى بها الجزء الأكبر من التحليلات في هذا الكتاب. إن النماذج التي تحتوي تراتبية الذاكرة أعقد فعلاً من نموذج RAM، ومن ثم يمكن أن يكون التعامل معها صعباً. إضافة إلى أن تحليلات نموذج RAM هي عادة مُتنبئات predictors متماثلة عن الأداء على الآلات الفعلية.

يمكن أن يشكل تحليل الخوارزمية - ولو كانت بسيطة - في نموذج RAM تحدياً؛ فقد تشمل الأدوات الرياضية اللازمة: التحليل التوافقي combinatorics، ونظرية الاحتمالات، وبراعة في الجبر، والقدرة على تحديد أهم المصطلحات في العبارة. ولما كان من الممكن أن يكون سلوك خوارزمية ما مختلفاً تبعاً لكل دخل ممكن، فإننا نحتاج إلى وسيلة لتلخيص هذا السلوك في عبارات بسيطة سهلة الفهم.

ومع أننا نختار عادة نموذج آلة واحدة فقط لتحليل خوارزمية ما، فنستغل نواجه خيارات كثيرة في تحديد كيفية التعبير عن تحليلنا. لذا فإننا نرغب في طريقة بسيطة للكتابة والمعالجة تُظهر للميزات الهامة لمتطلبات خوارزمية ما من الموارد، وتُجيبنا التفاصيل للملة.

### تحليل خوارزمية الفرز بالإدراج

يعتمد زمن تنفيذ إجراء الفرز بالإدراج على حجم الدخل: إذ إنَّ فَرْز ألف عددٍ يستغرق زمنًا أطول من الزمن المستغرق لفرز ثلاثة أعداد. إضافةً إلى ذلك، قد يستغرق الفرز بالإدراج أزمانًا مختلفة لفرز متتابعين دخل لهما الحجم نفسه اعتمادًا على مدى قرب وضعهما الابتدائي من الفرز المطلوب. يزداد زمن تنفيذ الخوارزمية عمومًا مع ازدياد حجم الدخل، لذلك يوصف زمن تنفيذ برنامج ما عادةً كدالةٍ لحجم دخله. ولفعل ذلك، نحتاج لأن نُعرِّف بعناية مصطلحي زمن التنفيذ وحجم الدخل.

يعتمد أفضل تعبير عن حجم الدخل *input size* على المسألة المدروسة. ففي أغلب المسائل، مثل: الفرز، أو حساب تحويلات فورييه المتقطعة، يكون القياس الطبيعي الأكثر هو عدد الحدود في الدخل. مثال ذلك، حجم الصيغة  $n$  في الفرز. وفي مسائل أخرى كثيرة، مثل ضرب عددين صحيحين، يكون القياس الأفضل لحجم الدخل هو العدد الكلي للبيانات اللازمة لتمثيل الدخل في التكوين الثنائي النظامي ordinary binary notation. وفي بعض الأحيان، يكون من الأنسب وصف حجم الدخل بعددين بدلًا من عدد واحد. فمثلًا، إذا كان دخل الخوارزمية يانًا graph، يمكن أن يوصف الدخل بعدد عقد vertices البيانات ووصلاته edges. سنبيّن بوضوح مقياس حجم الدخل المستخدم في كل مسألة ندرسها.

أما زمن تنفيذ *running time* خوارزمية ما على حجم دخل محدد، فهو عدد العمليات الأولية أو "الخطوات" steps المنفذة. من المناسب هنا تعريف فكرة الخطوة بحيث تكون مستقلة عن الآلة قدر الإمكان. سنعمد، مؤقتًا، الفكرة التالية: يحتاج كل سطرٍ من شبه رمزنا إلى مقدار ثابت من الزمن لتنفيذه. قد يستغرق سطرًا ما قدرًا من الزمن مختلفًا عما يستغرقه سطرٌ آخر، لكننا سنفترض أن كل تنفيذ للسطر ذي الرقم  $i$  يستغرق زمنًا  $c_i$ ، حيث  $c_i$  ثابت. نتسجم هذه الفكرة مع نموذج RAM، ونُظهر أيضًا كيف يمكن أن يُنحَتر شبه الرمز على معظم الحواسيب الفعلية.<sup>5</sup>

سوف يتطور في المناقشة التالية تعبيرنا عن زمن تنفيذ الفرز بالإدراج INSERTION-SORT من صيغة شائكة نستخدم جميع تكاليف التعليمات  $c_i$  إلى تكوين بسيط يكون أكثر إيجازًا وأسهل في المعالجة. وهذا التكوين البسيط سيجعل من السهل كذلك تحديد فعالية خوارزمية ما بالنسبة إلى خوارزمية أخرى. سنبدأ بعرض إجراء الفرز بالإدراج مع "الكلفة" الزمنية لكل عبارة وعدد مرات تنفيذ كل عبارة. في حال

<sup>5</sup> توجد هنا بعض النقاط الدقيقة؛ فالخطوات المحوسبة التي تحددها بالإنكليزية هي غالبًا أشكالًا متنوعة من إجراء يتطلب أكثر من مجرد مقدار ثابت من الزمن. فمثلًا، قد نستعمل في هذا الكتاب عبارة "افرز النقاط حسب الإحداثيات  $x$ "، وهذا، كما سترى، يستغرق أكثر من مقدار ثابت من الزمن. وكذلك، فإن الطليمة التي تستدعي مساقًا فرعيًا تستغرق زمنًا ثابتًا، مع أن المساق الفرعي، بمجرد استدعائه، قد يستغرق زمنًا أطول. وهذا يعني أننا نُفصل عملية استدعاء المساق الفرعي - أي تمرير الوسطاء إليه، إلخ - عن عملية تنفيذ المساق الفرعي.

حيث  $n = A.length$ ، نرمز بـ  $t_j$  لعدد مرات تنفيذ حلقة الاختيار **while** لقيمة  $j$  في السطر الخامس. عند الخروج من حلقة **while** أو حلقة **for** بالطريقة العادية (بسبب الاختيار في ترويسة الحلقة)، يتخذ الاختيار مرة واحدة زيادةً على عدد مرات تنفيذ جسم الحلقة. نفترض أن التعليقات ليست عبارات تنفيذية، لذلك لا تأخذ زمن تنفيذ.

INSERTION-SORT(A)	cost	times
1 <b>for</b> $j = 2$ <b>to</b> $A.length$	$c_1$	$n$
2 $key = A[j]$	$c_2$	$n - 1$
3     // Insert $A[j]$ into the sorted sequence $A[1 \dots j - 1]$ .	0	$n - 1$
4 $i = j - 1$	$c_4$	$n - 1$
5 <b>while</b> $i > 0$ and $A[i] > key$	$c_5$	$\sum_{j=2}^n t_j$
6 $A[i + 1] = A[i]$	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 $i = i - 1$	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 $A[i + 1] = key$	$c_8$	$n - 1$

إن زمن تنفيذ الخوارزمية هو مجموع أزمنة تنفيذ كل عبارة منفذة؛ فالعبارة التي تأخذ  $c_1$  خطوة لتنفيذها وتنفذ  $n$  مرة متساهم بمقدار  $c_1 n$  في زمن التنفيذ الكلي.<sup>6</sup> لحساب  $T(n)$ ، وهو زمن تنفيذ الفرز بالإدراج، لدخول مؤلف من  $n$  قيمة، نجمع جداء عمودي الكلفة والأزمنة، فينتج:

$$T(n) = c_1 n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$$

إذا كانت المدخلات من حجم معطى، فإن زمن تنفيذ الخوارزمية قد يعتمد على الدخول المغطى من ذلك الحجم. ففي INSERTION-SORT مثلاً، تحدث الحالة الفضلى إذا كانت الصفيغة مفروزة سابقاً. حيث نجد عندها أن  $A[i] \leq key$  لكل  $j = 2, 3, \dots, n$  في السطر الخامس عندما يأخذ المتحول  $i$  قيمته الابتدائية على أنها  $1 - j$ . وبهذا، تكون  $t_j = 1$  لكل  $j = 2, 3, \dots, n$  ويكون زمن التنفيذ في أفضل الحالات:

$$T(n) = c_1 n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1) \\ = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8) .$$

<sup>6</sup> لا تصح هذه الخاصية بالضرورة لمجرد كالتفكير. فالعبارة التي تتعامل مع  $m$  كلمة من الذاكرة وتنفذ  $n$  مرة لا تستهلك بالضرورة  $mn$  كلمة ذاكرة إجمالاً.

نستطيع أن نعبّر عن زمن التنفيذ هذا بالعلاقة  $an + b$  حيث  $a$  و  $b$  ثابتان يتعلّقان بكلف العبارات  $c_i$  وهي بذلك **دالة خطية Linear function** في  $n$ .

أما إذا كانت الصيغة مفروزة بترتيب عكسي - أي بترتيب متناقص - فنتجت عندها أسوأ الحالات. وعلينا عندها مقارنة كل عنصر  $A[j]$  بكل عنصر من داخل الصيغة الجزئية المفروزة  $A[1..j-1]$ ، وهكذا يكون  $t_j = j$  لكل  $j = 2, 3, \dots, n$ . وملاحظة أن:

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1$$

و

$$\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

(انظر الملحق أ لمراجعة كيفية حل هذه المجموع)، نجد أن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left( \frac{n(n+1)}{2} - 1 \right) \\ &\quad + c_6 \left( \frac{n(n-1)}{2} \right) + c_7 \left( \frac{n(n-1)}{2} \right) + c_8(n-1) \\ &= \left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8). \end{aligned}$$

ونستطيع أن نعبّر عن زمن تنفيذ أسوأ الحالات بالعبارة  $an^2 + bn + c$  حيث  $a$  و  $b$  و  $c$  ثوابت تتعلق أيضاً بكلف العبارات  $c_i$ ، وبهذا فهي **دالة من الدرجة الثانية quadratic function** في  $n$ .

وكما في الفرز بالإدراج، يكون زمن تنفيذ خوارزمية ما عادةً ثابتاً للدخل معطى، على الرغم من أننا سنجد في الفصول اللاحقة بعض الخوارزميات "ذات العشوائية المضادة" الممتعة التي يمكن أن يتغير سلوكها حتى في دخل ثابت.

### تحليل أسوأ الحالات والحالة الوسطى

تناولنا في تحليلنا للفرز بالإدراج كلاً من أفضل الحالات، التي تكون فيها صيغة الدخل مفروزة سلفاً، وأسوأ الحالات، التي تكون فيها صيغة الدخل مفروزة عكسياً. لكننا سنركز في بقية هذا الكتاب على إيجاد زمن تنفيذ أسوأ الحالات فقط، وذلك لأنه أطول زمن تنفيذ لأي دخل حجمه  $n$ . ونعطي هنا ثلاثة أسباب لهذا التوجّه.

- يعطينا زمن تنفيذ أسوأ الحالات لخوارزمية ما حلاً أعلى لزمن التنفيذ لأي دخل. وتعطينا معرفته ضماناً



أن الخوارزمية لن تستغرق زمناً أطول منه. حيث لا نحتاج إلى تخمين زمن التنفيذ آملين ألا يصبح أسوأ من ذلك التخمين.

- في بعض الخوارزميات، يتكرر ورود أسوأ الحالات مراراً؛ فمثلاً، أثناء البحث في قاعدة معطيات عن جزء محدد من المعلومات، سنحدث أسوأ الحالات لخوارزمية البحث كثيراً عندما لا تكون المعلومات موجودة في قاعدة للعطيات. وفي بعض التطبيقات، قد يكون البحث عن معلومات غير موجودة كثير الحدوث.
- كثيراً ما تكون "الحالة الوسطى" ماثلة - إلى حد ما - لأسوأ الحالات في السوء. لنفترض أننا اخترنا عشوائياً  $n$  عدداً، وأتينا طبقنا الغرز بالإدراج، فكم سنستغرق الخوارزمية لتحديد موضع إدراج عنصر ما  $A[j]$  في صيغة جزئية  $A[1..j-1]$ ؟ وسطياً، نصف العناصر في  $A[1..j-1]$  هي أقل من  $A[j]$ ، ونصف العناصر أكبر منه. لذا فإننا سنختار في المتوسط نصف عناصر الصيغة الجزئية  $A[1..j-1]$ ، وبذلك يساوي  $j/2$  تقريباً. وسيؤول زمن تنفيذ الحالة الوسطى الناتج إلى دالة من الدرجة الثانية في حجم الدخل، تماماً كما هو الحال في زمن تنفيذ أسوأ الحالات.

سوف نتم - في بعض الحالات الخاصة - بزمن تنفيذ الحالة الوسطى *average-case* للخوارزمية؛ وستتعرف نقانة التحليل الاحتمالي مطبقة على عدة خوارزميات في هذا الكتاب. من ناحية ثانية، إن مجال تحليل الحالة الوسطى محدود، لأنه قد لا يكون واضحاً ما هو الدخل الذي يمثل دخل "الحالة الوسطى" لمسألة محددة. سنفترض، في الغالب، أن احتمال حدوث جميع المدخلات من حجم معين متساوٍ. وعملياً، قد لا تكون هذه الفرضية محققة، لكن يمكننا في بعض الأحيان استخدام *خوارزمية ذات عشوائية مضافة* *randomized algorithm*، تقوم بعمليات عشوائية؛ لإتاحة التحليل الاحتمالي وحساب زمن تنفيذ متوقع *expected*. سندرس الخوارزميات ذات العشوائية للمضافة بتفصيل أكبر في الفصل الخامس وفي عدة فصول تالية أخرى.

### مقارنة النمو

لقد استخدمنا بعض التعريفات المبسطة لتسهيل تحليلنا لإجراء الغرز بالإدراج. من ذلك أننا تجاهلنا الكلفة الحقيقية لكل عبارة، وذلك باستخدام الثوابت  $c_i$  لتمثيل هذه الكلف. ثم لاحظنا أنه حتى هذه الثوابت تعطينا تفاصيل أكثر مما نحتاج إليه حقيقة؛ وعثرنا عن زمن تنفيذ أسوأ الحالات بالصيغة  $an^2 + bn + c$  لبعض الثوابت  $a$  و  $b$  و  $c$  التي تعتمد على كلف العبارات  $c_i$ . وبهذا لم نتجاهل فقط الكلف الحقيقية للعبارة، بل الكلف المجردة  $c_i$  أيضاً.

سنقوم الآن بتعريف مبسط إضافي: وهو *معدل نمو* *rate of growth* (أو *مرتبة نمو* *order of growth*) زمن التنفيذ الذي يهمنا حقيقة. ولهذا السبب، نتم فقط بالحد الرئيسي *leading term* في الصيغة

مثلاً  $an^2$ )، لأن الحدود ذات الدرجة الدنيا تكون غير مؤثرة نسبياً عند قيم  $n$  الكبيرة. كذلك، تحمل المعامل الثابت للحد الرئيسي، لأن العوامل الثابتة أقل تأثيراً من معدل النمو في تحديد الفعالية الحسابية للمدخلات الكبيرة. ففي حالة الفرز بالإدراج، عندما نتعامل الحدود ذات المرتبة الدنيا والمعامل الثابت الذي يسبق الحد الرئيسي، فإننا نُبقي على العامل من الدرجة  $n^2$  من الحد الرئيسي. ونقول إن للفرز بالإدراج زمن تنفيذ في أسوأ الحالات  $\Theta(n^2)$  (تلفظ "ثيتا  $n$  مربع"). نستخدم تدوين  $\Theta$  في هذا الفصل دون تعريف دقيق له، وسنعرّفه بدقة في الفصل ٣ .

نعتبر عادة أن خوارزمية ما أكثر فعالية من خوارزمية أخرى إذا كان زمن تنفيذها في أسوأ الحالات مرتبة نمو أدنى. لكن، قد تستغرق خوارزمية - زمن تنفيذها مرتبة نمو أعلى - زمناً أقل في حالة مدخلات أصغر من خوارزمية زمن تنفيذها مرتبة نمو أدنى، وذلك بسبب العوامل الثابتة والحدود الأدنى مرتبة. إلا أنه، في حالة مدخلات كبيرة كفاية، ستنفذ الخوارزمية ذات الـ  $\Theta(n^2)$  مثلاً، بسرعة أكبر في أسوأ الحالات من خوارزمية المرتبة  $\Theta(n^3)$ .

## تمارين

### 1-2.2

عبر عن الدالة  $3 - 100n + 100n^2 - n^3/1000$  باستخدام التدوين  $\Theta$ .

### 2-2.2

لنكن لدينا مسألة فرز  $n$  عددًا محزناً في صيغة  $A$ ، وذلك بأن نوجد أولاً أصغر عنصر في  $A$ ، ونبادله مع العنصر  $A[1]$ ، ثم نوجد ثاني أصغر عنصر في  $A$ ، ونبادله مع العنصر  $A[2]$ . ونستمر بهذه الطريقة للعناصر الـ  $n - 1$  الأولى من  $A$ . اكتب شبه رماز لهذه الخوارزمية، التي تُعرف بالفرز الانتقالي *selection sort*. ما هو لامتغر الحلقة الذي تصونه هذه الخوارزمية؟ لماذا يجب أن تنفذ فقط من أجل الـ  $n - 1$  عنصرًا الأولى من الصيغة؟ أعط أزمّة تنفيذ الفرز الانتقالي في أفضل الحالات، وفي أسوأ الحالات باستخدام تدوين  $\Theta$ .

### 3-2.2

ليكن لدينا البحث الخطي ثانية (انظر التمرين 3-1.2). ما هو عدد العناصر التي يجب اختبارها وسطياً من متتالية الدخل، بافتراض أن العنصر الذي نبحث عنه قد يكون أيّاً من عناصر في الصيغة باحتمال متساوٍ؟ وما هو عدد العناصر التي يجب اختبارها في أسوأ الحالات؟ ما هو زمن تنفيذ البحث الخطي في الحالة الوسطى، وفي أسوأ الحالات باستخدام تدوين  $\Theta$ ؟ علّل أجوبتك.

### 4-2.2

كيف يمكننا تعديل - تقريباً - أية خوارزمية للحصول على زمن تنفيذ جيد للحالة الفضلى؟

## 3.2 تصميم الخوارزميات

نستطيع الاختيار ضمن طيف واسع من تقنيات تصميم الخوارزميات. فقد استخدمنا في حالة خوارزمية الفرز بالإدراج طريقة *تزايدية incremental*: فبعد فرز الصفحة الجزئية  $A[1..j-1]$ ، أدرجنا العنصر  $A[j]$  في مكانه المناسب، لتنتج الصفحة الجزئية المقررة  $A[1..j]$ .

سنطرق في هذا المقطع إلى طريقة تصميم بديلة تدعى "فرق-تسُد divide-and-conquer"، سنعرضها بتفصيل أكبر في الفصل 4. سنستخدم مفهوم فرق-تسد لتصميم خوارزمية فرز زمن تنفيذها في أسوأ الحالات أقل كثيرًا من زمن تنفيذ الفرز بالإدراج في أسوأ الحالات. إحدى فوائد خوارزميات فرق-تسد أنه يمكن غالبًا تحديد أزمته تنفيذها بسهولة باستخدام تقنيات سنراها في الفصل 4.

### 1.3.2 طريقة فرق-تسُد

كثير من الخوارزميات المفيدة *عُودبة recursive* في بنيتها: فحلّ مسألة معطاة، تستدعي هذه الخوارزميات نفسها عوديًا مرة أو أكثر لمعالجة مسائل جزئية ذات علاقة وثيقة بالمسألة الأصلية. تُشغّل هذه الخوارزميات في أغلب الأحيان طريقة *فرق-تسُد divide-and-conquer*: حيث تُقسّم المسألة الأصلية إلى عدة مسائل جزئية تشابه المسألة الأصلية لكنها أصغر حجمًا، ثم تُحلّ هذه المسائل الجزئية عوديًا، ثم تُجمّع هذه الحلول لتكوين حل المسألة الأصلية.

يحتوي نموذج فرق-تسُد على ثلاث خطوات في كل مستوى من العُودبة:

**فرق:** قسّم المسألة إلى عدد من المسائل الجزئية التي هي متسَخاتّ instances أصغر من المسألة نفسها. **سُد:** سَيطر على المسائل الجزئية بحلّها عوديًا. فإذا أصبحت حجوة المسائل الجزئية صغيرة كفاية، فحلّ المسائل الجزئية مباشرة.

**جمّع:** جَمّع حلول المسائل الجزئية لتكوين حلّ المسألة الأصلية.

تُشغّل خوارزمية **الفرز بالدمج Merge Sort** مبدأ فرق-تسُد تمامًا؛ فهذه الخوارزمية تعمل ببساطة كما يلي:

**فرق:** قسّم المتتالية المكونة من  $n$  عنصراً المطلوب فرزها إلى متتاليتين جزئيتين، يتكوّن كل منهما من  $n/2$  عنصراً.

**سُد:** افرز المتتاليتين الجزئيتين عوديًا باستخدام الفرز بالدمج.

**جمّع:** ادمج المتتاليتين الجزئيتين المقرورتين لإنتاج الجواب للفرز.

تنتهي العملية القودية عندما يصبح طول للتالية المطلوب فرزها يساوي 1، حيث لا يوجد ما يجب فعله في هذه الحالة، لأن كل متتالية طولها 1 تكون بترتيب مفروز حُكْمًا.

العملية الأساسية في خوارزمية الفرز بالدمج هي دمج متالتين مفروزتين في خطوة "التجميع". ندمج باستدعاء إجراء مساعد  $MERGE(A, p, q, r)$  حيث  $A$  صفيقة، و  $p$ ، و  $q$ ، و  $r$  دلائل في الصفيقة بحيث يكون  $p \leq q < r$ . يفترض الإجراء أن الصفيقتين الجزئيتين  $A[p..q]$  و  $A[q+1..r]$  مفروزان. يدمجهما  $merges$  هذا الإجراء ليكون صفيقة جزئية وحيدة مفروزة تحمل محل الصفيقة الجزئية الحالية  $A[p..r]$ .

يستغرق الإجراء  $MERGE$  زمنًا  $\Theta(n)$ ، حيث  $n = r - p + 1$  هو عدد العناصر الكلي التي يجري دمجها، ويعمل الإجراء كما يلي: بالعودة إلى تصورنا عن لعبة ورق الشدة، لكن لدينا على الطاولة كومتين من أوراق الشدة وجوهها إلى أعلى. كل كومة مفروزة، بحيث تكون أصغر الأوراق في أعلى الكومة. نريد دمج هاتين الكومتين في كومة نخرج واحدة مفروزة، بحيث تكون وجوه الأوراق فيها مقلوبة للأسفل على الطاولة. تتألف خطواتنا الأساسية من اختيار أصغر الورقتين للموضوعتين على قمتي كومتنا الأوراق التي وجوهها إلى الأعلى؛ ثم إزالتها من كومتها (وهذا يؤدي إلى كشف ورقة قمة جديدة)، ووضع هذه الورقة على كومة الخرج وجوهها مقلوب للأسفل. نكرر هذه الخطوة حتى تصبح إحدى كومتنا الدخول فارغة، عندها فقط نأخذ كومة الدخول المتبقية ونضعها على كومة الخرج وجوهها إلى الأسفل. حسابيًا، تستغرق كل خطوة أساسية زمنًا ثابتًا، لأننا نقوم فقط بمقارنة ورقتي قمة. ولما كنا ننفذ  $n$  خطوة أساسية على الأكثر، فإن عملية الدمج تستغرق زمنًا  $\Theta(n)$ .

يُنحَرُ شبه الرماز التالي الفكرة المذكورة آنفًا، لكن مع تعديل إضافي يجنبنا الحاجة إلى التحقق من كون إحدى الكومتين فارغة في كل خطوة أساسية. نضع في أسفل كل كومة ورقة خاصة نسميها *الورقة الكاشفة sentinel card*، تحتوي قيمة خاصة نستخدمها لتبسيط رمازنا. سنستخدم  $\infty$  قيمة كاشفة، بحيث أنه حالما نكشف الورقة ذات القيمة  $\infty$ ، فلا يمكن أن تكون الورقة الصغرى ما لم تُظهر كلتا الكومتين وترتيبهما الكاشفتين. لكن ما إن يحدث ذلك، تكون جميع الأوراق الأخرى قد وُضعت على كومة الخرج. ولأننا نعرف سلفًا أنه متوضّع بالضغط  $r - p + 1$  ورقة على كومة الخرج، فإننا نستطيع التوقف حالما نكون قد قمنا بهذا القدر من الخطوات الأساسية.

$MERGE(A, p, q, r)$

- 1  $n_1 = q - p + 1$
- 2  $n_2 = r - q$
- 3 let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
- 4 for  $i = 1$  to  $n_1$
- 5  $L[i] = A[p + i - 1]$

```

6  for j = 1 to n2
7      R[j] = A[q + j]
8  L[n1 + 1] = ∞
9  R[n2 + 1] = ∞
10 i = 1
11 j = 1
12 for k = p to r
13     if L[i] ≤ R[j]
14         A[k] = L[i]
15         i = i + 1
16     else A[k] = R[j]
17         j = j + 1

```

يُعمل إجراء الدمج MERGE بالتفصيل كما يلي: يُحسب السطر الأول المقدار  $n_1$  وهو طول الصيغة الجزئية  $A[p..q]$ ، ويُحسب السطر الثاني المقدار  $n_2$  وهو طول الصيغة الجزئية  $A[q + 1..r]$ . ننشئ في السطر الثالث الصيغتين  $L$  و  $R$  ("الصيغة اليسرى" و "الصيغة اليمنى")، طول الأولى  $n_1 + 1$ ، وطول الثانية  $n_2 + 1$ ؛ وسيحتوي الموقع الإضافي في كل صيغة الورقة الكاشفة. ننسخ حلقة **for** المكونة من السطرين 4-5 الصيغة الجزئية  $A[p..q]$  في  $L[1..n_1]$ ، وننسخ حلقة **for** المكونة من السطرين 6-7 الصيغة الجزئية  $A[q + 1..r]$  في  $R[1..n_2]$ . يُنسخ السطران 8-9 الورتين الكاشفتين في نهايتي الصيغتين  $L$  و  $R$ . تُنسخ الأسطر 10-17، الموضحة في الشكل 3.2، الـ  $r - p + 1$  خطوة أساسية بالمحافظة على لامتغير الحلقة التالي:

عند بداية كل تكرار من حلقة **for**، التي تشمل الأسطر 12-17، تحتوي الصيغة الجزئية  $A[p..k-1]$  أصغر  $k - p$  عنصراً من  $L[1..n_1 + 1]$  و  $R[1..n_2 + 1]$ ، بترتيب مفروز. إضافة إلى ذلك، يكون العنصران  $L[i]$  و  $R[j]$  أصغر عنصرين في صيغتيهما لم ينسخا إلى  $A$ .

يجب أن نُثبت أن لامتغير الحلقة هذا محقق قبل التكرار الأول من حلقة **for** التي تشمل الأسطر 12-17، وأن كل تكرار من الحلقة يحافظ على اللامتغير، وأن لامتغير الحلقة هذا يوفر خاصية مفيدة لإظهار صحة العمل عند توقف الحلقة.

الاستدعاء: لدينا  $k = p$  قبل التكرار الأول للحلقة، أي إن الصيغة الجزئية  $A[p..k-1]$  فارغة. تحتوي هذه الصيغة الجزئية الفارغة أصغر  $k - p = 0$  عنصراً من  $L$  و  $R$ ، ولما كان  $i = j = 1$ ، فإن كلاً من العنصرين  $L[i]$  و  $R[j]$  هما أصغر عنصرين في صيغتيهما لم يُنسخا في  $A$ .

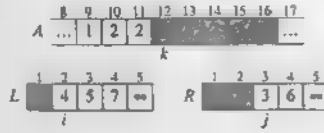
المحافظة: حتى نرى أن كل تكرار يحافظ على لامتغير الحلقة، لنفترض أولاً أن  $L[i] \leq R[j]$ . عندها يكون  $L[i]$  هو أصغر عنصر لم يُنسخ نُسخه يُنقذ إلى  $A$ . ولما كانت  $A[p..k-1]$  تحتوي أصغر  $k - p$  عنصراً،



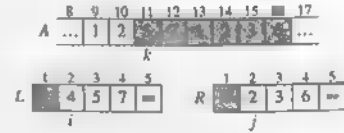
(ب)



(أ)



(ث)



(د)

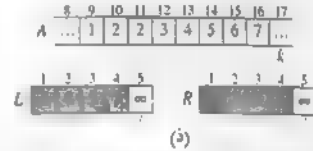
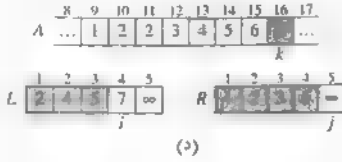
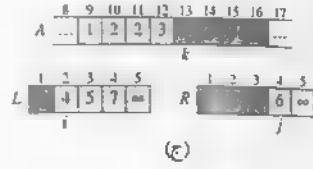
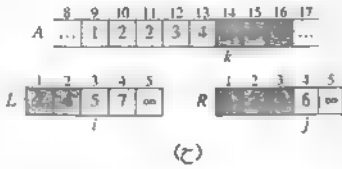
**الشكل 3.2** عمل الأسطر 10-17 في الاستدعاء  $\text{MERGE}(A, 9, 12, 16)$ ، عندما تحتوي الصفيفة الجزئية  $A[9..16]$  المتتالية  $\{2, 4, 5, 7, 1, 2, 3, 6\}$ . بعد النسخ وإدخال القيمتين الكاشفتين، ستحتوي الصفيفة  $L$  القيم  $\{2, 4, 5, 7, \infty\}$ ، وستحتوي الصفيفة  $R$  القيم  $\{1, 2, 3, 6, \infty\}$ . تحتوي المواضع المظللة باللون الفاتح في  $A$  قيمها النهائية، وتحتوي المواضع المظللة باللون الفاتح في  $L$  و  $R$  قيمًا يجب أن يعاد نسخها إلى  $A$ . بأخذ المواضع المظللة باللون الفاتح مقًا، تحتوي هذه المواضع دائمًا القيم الموجودة أصلاً في  $A[9..16]$ ، إضافة إلى القيمتين الكاشفتين. وتحتوي المواضع المظللة بكثافة في  $A$  قيمًا سوف يتم النسخ فوقها، وتحتوي المواضع المظللة بكثافة في  $L$  و  $R$  قيمًا تم إعادة نسخها إلى  $A$ . الأشكال من (أ) إلى (د) تمثل الصفيفات  $A$  و  $L$  و  $R$  التي دللناها  $k$  و  $i$  و  $j$  على الترتيب، قبل كل تكرار من الحلقة في الأسطر 17-12.

فبعد أن ينسخ السطر 14 العنصر  $L[i]$  إلى  $A[k]$ ، ستحتوي الصفيفة الجزئية  $A[p..k]$  أصغر  $k-p+1$  عنصرًا. تعيد زيادة كل من  $k$  (أثناء تحديث حلقة **for**) و  $i$  (في السطر 15) إعداد لامتغير الحلقة للتكرار التالي. أما في حال  $L[i] > R[j]$ ، فإن السطرين 16-17 تقوم بما يجب للمحافظة على لامتغير الحلقة.

**الإنهاء:** عند الإنهاء، يكون  $k = r + 1$ . واعتمادًا على لامتغير الحلقة، تحتوي الصفيفة الجزئية  $A[p..k-1]$  التي هي  $A[p..r]$ ، أصغر  $k-p = r-p+1$  عنصرًا من  $L[1..n_1+1]$  و  $R[1..n_2+1]$  بترتيب مفروز. أي تحتوي الصفيفتان  $L$  و  $R$  معًا  $n_1 + n_2 + 3 = r-p+3$  عنصرًا. كلها قد أعيد نسخها في الصفيفة  $A$ ، عدا أكبر عنصرين، وهما القيمتان الكاشفتان.

حتى ترى أن إجراء الدمج **MERGE** يُنفذ في زمن  $\Theta(n)$ ، حيث  $n = r-p+1$ ، لاحظ أن كلاً من الأسطر 3-1 و 8-11 يستغرق زمنًا ثابتًا، وتستغرق حلقات الـ **for** في الأسطر 7-4 زمنًا قدره  $\Theta(n_1 + n_2) = \Theta(n)$ <sup>7</sup>، وأن هناك  $n$  تكرارًا لحلقة **for** في الأسطر 12-17، يستغرق كل تكرار منها زمنًا ثابتًا.

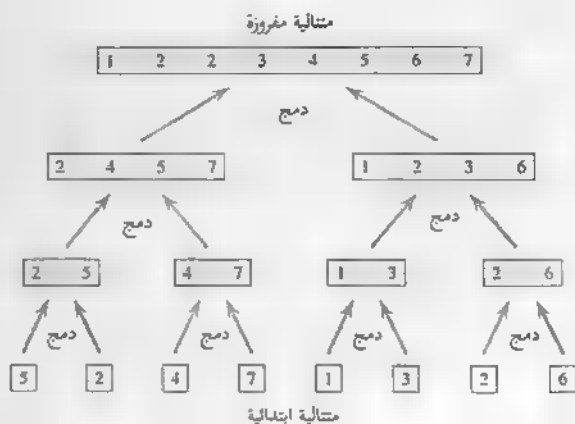
<sup>7</sup> سنرى في الفصل الثالث كيف تفسر المعادلات التي تحتوي على تلوين  $\Theta$  صورًا.



يُتبع الشكل 3.2 يُمثل الشكل (ذ) الصفقات والمؤشرات عند الانتهاء. عند هذه المرحلة، تصبح الصفقة الجزئية  $A[9..16]$  مفروزة، والقيمتان الكاشفتان في  $L$  و  $R$  هما العنصران الوحيدان في هاتين الصفقتين اللذان لم ينسجما بقُد إلى  $A$ .

نستطيع الآن استخدام إجراء الدمج MERGE كمسألة فرعية في خوارزمية الفرز بالدمج. بفرض الإجراء  $\text{MERGE-SORT}(A, p, r)$  عناصر الصفقة الجزئية  $A[p..r]$ . إذا كان  $p \geq r$ ، تحتوي الصفقة الجزئية على الأكثر عنصرًا واحدًا وتكون مفروزة أصلاً. وفي الحالات الأخرى، نحسب خطوات التقسيم ببساطة دليلاً  $q$  ينقسم  $A[p..r]$  إلى صفقتين فرعيتين: الأولى:  $A[p..q]$ ، وهي تحوي  $\lfloor n/2 \rfloor$  عنصرًا، والثانية:  $A[q+1..r]$ ، وهي تحوي  $\lfloor n/2 \rfloor$  عنصرًا. ■

■ تعني العبارة  $\lfloor x \rfloor$  أصغر عدد صحيح أكبر من  $x$  أو يساويه، وتعني العبارة  $\lceil x \rceil$  أكبر عدد صحيح أقل من  $x$  أو يساويه. هذه الترميزات معروفة في الفصل 3. إن أسهل طريقة للتحقق من أن إعطاء  $q$  القيمة  $\lfloor (p+r)/2 \rfloor$  ينتج الصفقتين الفرعيتين  $A[p..q]$  و  $A[q+1..r]$  وطولهما  $\lfloor n/2 \rfloor$  و  $\lfloor n/2 \rfloor$  على الترتيب، هو اختبار الحالات الأربع التي تظهر فيها لكون كل من  $p$  و  $r$  فرديًا أو زوجيًا.



**الشكل 4.2** عمل الفرز بالدمج على الصيغة  $A = \{5, 2, 4, 7, 1, 3, 2, 6\}$ . تزداد أطوال المتتاليات المفروزة التي يتم دمجها كلما تقدمت الخوارزمية من الأسفل إلى الأعلى.

**MERGE-SORT( $A, p, r$ )**

```

1  if  $p < r$ 
2       $q = \lfloor (p + r) / 2 \rfloor$ 
3      MERGE-SORT( $A, p, q$ )
4      MERGE-SORT( $A, q + 1, r$ )
5      MERGE( $A, p, q, r$ )

```

لفرز كامل المتتالية  $(A[1], A[2], \dots, A[n])$  نقوم بالاستدعاء البدئي  $\text{MERGE-SORT}(A, 1, A.length)$ ، حيث  $A.length = n$ . يوضح الشكل 4.2 كيفية عمل الإجراء من الأسفل إلى الأعلى عندما تكون  $n$  من قوى 2. تنطوي الخوارزمية على دمج أزواج من متتاليات ذات عنصر وحيد لتكوّن متتاليات مفروزة طول كل منها 2، ثم على دمج أزواج من المتتاليات ذات الطول 2 لتكوّن متتاليات مفروزة طول كل منها 4، وهكذا حتى تُدمج متتاليتان طول كل منهما  $n/2$  لتكوّن المتتالية النهائية المفروزة وطولها  $n$ .

### 2.3.2 تحليل خوارزميات فرق-تشد

عندما تحتوي خوارزمية ما استدعاءً عودياً، يكون بمقدورنا غالباً وصف زمن تنفيذها **بمعادلة عودية** *recurrence equation* أو **بمعادلة عودية** *recurrence*. نصف الزمن الكلي لمسألة من الحجم  $n$  بدلالة زمن التنفيذ على مدخلات أصغر. نستطيع عندها استخدام أدوات رياضية لحل العلاقة العودية وإعطاء حدود لأداء الخوارزمية.

تنتج العلاقة العودية لزمن تنفيذ خوارزمية فرق-تشد من الخطوات الثلاث التي تكوّن إطار العمل



الأساسي. كما ذكرنا سابقاً، نعمل  $T(n)$  زمن التنفيذ لمسألة حجمها  $n$ . إذا كان حجم المسألة صغيراً كفايةً، وليكن  $c \leq n$  حيث  $c$  ثابت ما، يستغرق الحل المباشر زمناً ثابتاً، نكتبه  $\Theta(1)$ . لنفترض أن تقسيمنا للمسألة ينتج  $a$  مسألة جزئية، حجم كل منها يساوي  $1/b$  من حجم للمسألة الأصلية. (في الفرز بالدمج، يساوي كل من  $\blacksquare$  و  $b$  القيمة 2، لكننا سنرى كثيراً من خوارزميات فرق-تسد يكون فيها  $a \neq b$ ). يستغرق حل مسألة جزئية واحدة حجمها  $n/b$  زمناً  $T(n/b)$ ، وهكذا تستغرق  $a$  مسألة جزئية من هذا الحجم زمناً  $aT(n/b)$ . إذا استغرقتنا زمناً  $D(n)$  لتقسيم المسألة إلى مسائل جزئية، وزمناً  $C(n)$  لتكوين حلول المسائل الجزئية للحصول على حل للمسألة الأصلية، فإننا نحصل على العلاقة القودية الآتية:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n \leq c, \\ aT(n/b) + D(n) + C(n) & \text{otherwise.} \end{cases}$$

سنرى في الفصل 4 كيفية حل العلاقات القودية الشائعة من هذه الصيغة.

### تحليل الفرز بالدمج

على الرغم من أن شبه رماز خوارزمية MERGE-SORT يعمل على الوجه الصحيح عندما لا يكون عدد العناصر زوجياً، فإنه يمكن تبسيط تحليلنا المبني على العلاقة القودية إذا افترضنا أن حجم المسألة الأصلية هو من قوى 2. عندها ننتج كل خطوة تقسيم متاليتين جزئيتين حجم كل منهما  $n/2$  تماماً. سنرى في الفصل 4، أن هذه الفرضية لا تؤثر على مرتبة نمو حل العلاقة القودية.

ستتم المحاكمة المنطقية التالية لاستنتاج العلاقة القودية التي يحققها  $T(n)$ ، زمن تنفيذ خوارزمية الفرز بالدمج على  $n$  عدداً في أسوأ الحالات. يستغرق تنفيذ الفرز بالدمج على عنصر واحد فقط زمناً ثابتاً. وعندما يكون لدينا عدد العناصر  $n > 1$ ، نُجزئُ زمن التنفيذ كما يلي:

لوقت: نحسب خطواته التقريبي منتصف الصيغة الجزئية فقط، وهذا يستغرق زمناً ثابتاً. ومن ثم يكون:

$$D(n) = \Theta(1)$$

سُد: نحل عددياً مسألتين فرعيتين، حجم كل منهما  $n/2$ ، فيسهم هذا العمل في زمن التنفيذ بمقدار  $2T(n/2)$ .

جَمْع: لاحظنا تَوّاً أن إجراء الدمج MERGE المطبق على صيغة جزئية حجمها  $n$  عنصراً يستغرق زمناً  $\Theta(n)$ ، وبذلك يكون  $C(n) = \Theta(n)$ .

عندما نجمع الدالتين  $D(n)$  و  $C(n)$  في تحليل الفرز بالدمج، فإننا نجمع دالة نموه  $\Theta(n)$  مع دالة نموه  $\Theta(1)$ . إن هذا المجموع هو دالة خطية في  $n$ ، أي  $\Theta(n)$ . بالإضافة هذه الدالة إلى الحد  $2T(n/2)$  الناتج من خطوة "السيادة conquer" نتيج العلاقة القودية لـ  $T(n)$  زمن تنفيذ الفرز بالدمج في أسوأ الحالات:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases} \quad (1.2)$$

سنرى، في الفصل 4، "المبرهنة الرئيسة master theorem"، التي يمكننا استخدامها لبيان أن  $T(n)$  يساوي  $\Theta(n \lg n)$ ، حيث  $\lg n$  هو  $\log_2 n$ . ولما كان نمو الدالة اللغاريتمية أكثر بطئاً من أية دالة خطية، فإنه في حالة مدخلات كبيرة كفاية، يتفوق الفرز بالدمج ذي زمن التنفيذ  $\Theta(n \lg n)$  على الفرز بالإدراج الذي زمن تنفيذه  $\Theta(n^2)$ ، في أسوأ الحالات.

لا نحتاج إلى المبرهنة الرئيسة لنفهم بالحدس لماذا حل العلاقة القودية (1.2) هو  $T(n) = \Theta(n \lg n)$ . سنعيد كتابة العلاقة القودية (1.2) كما يلي:

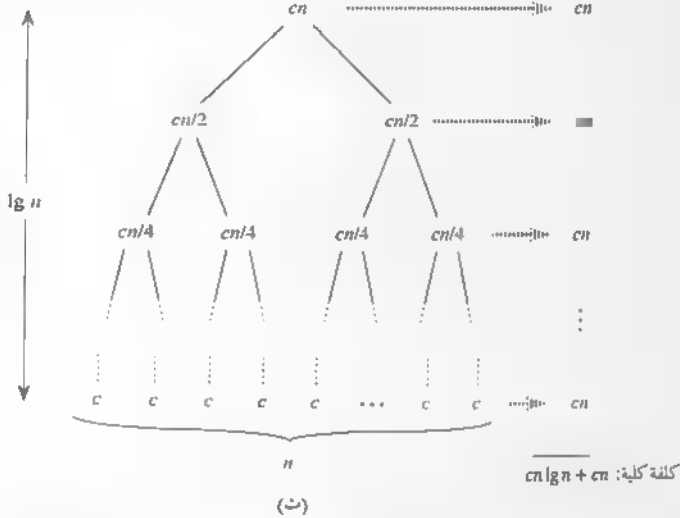
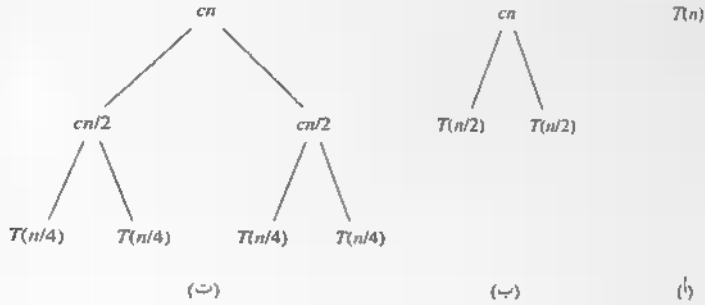
$$T(n) = \begin{cases} c & \text{if } n = 1, \\ 2T(n/2) + cn & \text{if } n > 1. \end{cases} \quad (2.2)$$

حيث يمثل الثابت  $c$  الزمن اللازم لحل مسائل من الحجم 1، ويمثل أيضاً زمن معطوبي التفريق والتجميع لكل عنصر من عناصر المصفوفة.<sup>9</sup>

يبين الشكل 5.2 كيف يمكننا حل العلاقة القودية (2.2). نفترض للتبسيط، أن  $n$  هو بالضبط قوة صحيحة من قوى العدد 2. يبين الجزء (أ) من الشكل للقدار  $T(n)$ ، الذي نشره في الجزء (ب) إلى شجرة مكافئة تمثل العلاقة القودية. الحد  $cn$  هو الجذر (وهو الكلفة المتضمنة عند المستوى الأعلى من القودية)، والشجرتان الفرعيتان للجذر هما العلاقتان القوديتان الصغريان  $T(n/2)$ . يظهر الجزء (ت) إنجاز خطوة إضافية من هذه العملية بنشر  $T(n/2)$ . أما الكلفة المتضمنة عند كل من العقدتين الفرعيتين في المستوى الثاني من القودية فهي  $cn/2$ . نتابع نشر كل عقدة من الشجرة بتقسيمها إلى أجزائها المكونة كما هو محدد في العلاقة القودية، حتى تنخفض أحجام المسألة إلى 1، حيث تكون كلفة كل منها  $c$ . يبين الجزء (ث) شجرة القودية *recursion tree* الناتجة.

بعد ذلك، نجمع الكلف عند كل مستوى من الشجرة. للمستوى الأعلى كلفة كلية تساوي  $cn$ ، والكلفة الإجمالية للمستوى الأدنى التالي هي  $c(n/2) + c(n/2) = cn$ ، والكلفة الكلية للمستوى الذي يليه هي  $cn = c(n/4) + c(n/4) + c(n/4) + c(n/4)$ ، وهكذا دواليك. عمومًا، يمتلك المستوى  $i$  الأدنى بدءًا من قمة الشجرة  $2^i$  عقدة، وتشارك كل عقدة منها بمقدار  $c(n/2^i)$  من الكلفة، بحيث يكون للمستوى  $i$  الأدنى بدءًا من القمة كلفة كلية تساوي  $cn = 2^i c(n/2^i)$ . يمتلك المستوى الأدنى الأخير  $n$  عقدة، وتشارك

<sup>9</sup> من غير المحتمل أن يمثل الثابت نفسه بدقة كلاً من زمن حل مسائل من الحجم 1 وزمن عنصر المصفوفة لخطوات التفريق والتجميع. يمكن أن نتجنب هذه المشكلة إما بأن نجعل  $c$  أكبر هذه الأربعة، مع إدراكنا أن تكرار القودية يعطي حلاً أعلى لزمن التنفيذ، وإما بأن نجعل  $c$  أقل هذه الأربعة، مع إدراكنا أن تكرار القودية يعطي الحد الأدنى لزمن التنفيذ. على أن كلا الحدين سيكون  $n \lg n$ ، وبأحدهما معاً يكون زمن التنفيذ  $\Theta(n \lg n)$ .



**الشكل 5.2** كيفية بناء شجرة القودية للعلاقة القودية  $T(n) = 2T(n/2) + cn$ . بين الجزء (أ)  $T(n)$  الذي يمتد تدريجيًا في الأجزاء (ب)-(ث) لبناء شجرة القودية. تمتلك شجرة القودية المنشورة كاملة في الجزء (ث)  $\lg n + 1$  مستوى (أي ارتفاعها  $\lg n$ ، كما هو مشار إليه)، ويشارك كل مستوى في الكلفة الكلية بالمقدار  $cn$ . وبناء على ذلك تساوي كلفة الزمن الكلية للمقدار  $cn \lg n + cn$  التي هي  $\Theta(n \lg n)$ .

كل منها في الكلفة بمقدار  $c$ ، وتكون الكلفة الكلية لهذا المستوى هي  $cn$ . إن العدد الكلي لمستويات الشجرة القودية recursion tree في الشكل 5.2 هو  $\lg n + 1$ ، حيث  $n$  عدد الأوراق، وهو يقابل حجم الدخل. ويمكن برهان هذا الادعاء بمحاكمة استقرائية مبسطة. تحدث الحالة

الأساسية عندما تكون  $n = 1$ ، في هذه الحالة تمتلك الشجرة مستوى واحدًا فقط. ولما كان  $\lg 1 = 0$ ، فإن  $\lg n + 1$  يعطي عدد المستويات الصحيح. لنفترض الآن فرضية استقرائية هي أن عدد مستويات الشجرة القودية لـ  $2^i$  ورقة هو  $i + 1$  (لأنه في حالة أية قيمة  $i$  نحصل على  $i = \lg 2^i$ ). وحيث إننا افترضنا أن حجم الدخول هو أحد قوى العدد 2، فإن حجم الدخول التالي هو  $2^{i+1}$ . تمتلك الشجرة التي تحتوي  $n = 2^{i+1}$  ورقة مستوى إضافيًا واحدًا مقارنة بعدد المستويات التي تمتلكها الشجرة التي عدد أوراقها  $2^i$ ، وهكذا يكون العدد الكلي للمستويات هو  $\lg 2^{i+1} + 1 = (i + 1) + 1$ .

لحساب الكلفة الكلية المثلثة بالعلاقة القودية (2.2)، نجمع ببساطة كلف جميع المستويات من الأسفل إلى الأعلى. ولما كانت شجرة القودية تمتلك  $\lg n + 1$  مستوى، كلفة كل منها  $cn$ ، فالكلفة الكلية تساوي  $cn(\lg n + 1) = cn \lg n + cn$ . ويتجاهل الحد ذي الدرجة الأدنى والثابت  $c$  نحصل على النتيجة المطلوبة وهي:  $\Theta(n \lg n)$ .

## تمارين

### 1-3.2

باستخدام الشكل 4.2 غودجًا، وضح عمل الفرز بالدمج على الصيغة التالية:  
 $A = \{3, 41, 52, 26, 38, 57, 9, 49\}$

### 2-3.2

أعد كتابة إجراء الـ MERGE بحيث لا يستخدم الأوراق الكاشفة، واجعله بدلاً عن ذلك يتوقف بمجرد أن تكون أيٌّ من الصفتين  $L$  أو  $R$  قد أعادت نسخ جميع عناصرها إلى الصيغة  $A$ ، ثم يقوم بإعادة نسخ العناصر المتبقية من الصيغة الأخرى إلى  $A$ .

### 3-3.2

استخدم الاستقراء الرياضي لإظهار أنه عندما تكون  $n$  قوة صحيحة للعدد 2، فإن حل العلاقة القودية التالية:

$$T(n) = \begin{cases} 2 & \text{if } n = 1, \\ 2T(n/2) + n & \text{if } n = 2^k, \text{ for } k > 1 \end{cases}$$

هو  $T(n) = n \lg n$ .

### 4-3.2

يمكننا التعبير عن الفرز بالإدراج كإجراء عودي كما يلي: لفرز  $A[1..n]$ ، نفرز عوديًا  $A[1..n-1]$ ، ثم نُدرج  $A[n]$  في الصيغة المرتبة  $A[1..n-1]$ . اكتب العلاقة القودية لزمن التنفيذ في أسوأ الحالات لهذه النسخة العودية من الفرز بالإدراج.

### 5-3.2

بالرجوع إلى مسألة البحث (انظر التمرين 3-1.2)، لاحظ أنه إذا كانت للمتتالية  $A$  مفروزة، نستطيع اختبار نقطة وسط هذه المتتالية مقارنة بالقيمة  $v$  وإخراج نصف للمتتالية من دائرة بحثنا التالي. تكرر خوارزمية البحث الثنائي  $binary\ search$  هذا الإجراء، وبذلك تنصّف حجم الجزء الباقي من المتتالية في كل مرة. اكتب شبه رماز للبحث الثنائي، إما تكرارياً وإما غُودياً. يَبَيّن أن زمن تنفيذ البحث الثنائي في أسوأ الحالات هو  $\Theta(\lg n)$ .

### 6-3.2

لاحظ أن حلقة **while** المكونة من الأسطر 7-5 من إجراء INSERTION-SORT في المقطع 1.2 تستخدم البحث الخطي لمسح الصفيفة الجزئية للمفروزة  $A[1..j-1]$  عكسياً. هل نستطيع استخدام البحث الثنائي (انظر التمرين 5-3.2) عوضاً عن تحسين زمن تنفيذ الفرز بالإدراج الكلي في أسوأ الحالات ليصبح  $\Theta(n \lg n)$ ؟

### \* 7-3.2

صِف خوارزمية ذات زمن تنفيذ قدره  $\Theta(n \lg n)$  تقوم - عند إعطائها مجموعة  $S$  مؤلفة من  $n$  عدداً صحيحاً وعدد صحيح آخر  $x$  - بتحديد وجود (أو عدم وجود) عنصرين في  $S$  مجموعتهما هو  $x$  تماماً.

## مسائل

### 1-2 الفرز بالإدراج على صفيفات صغيرة داخل الفرز بالدمج

على الرغم من أن زمن تنفيذ الفرز بالدمج في أسوأ الحالات هو  $\Theta(n \lg n)$ ، وزمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو  $\Theta(n^2)$ ، فإن العوامل الثابتة في الفرز بالإدراج يمكن أن تجعله أسرع عملياً في حالة أحجام المسائل الصغيرة على العديد من الحواسيب. وهذا ما قد يجعل مجزاً **تسليك** *coarsen* الأوراق في الغودية باستخدام الفرز بالإدراج ضمن الفرز بالدمج عندما تصبح للمسائل الجزئية صغيرة كافية. لندرس تعديلاً للفرز بالدمج نُفرز فيه لوائح عددها  $n/k$ ، طول كل منها  $k$  باستخدام الفرز بالإدراج، ومن ثم تدمج باستخدام آلية الدمج للمعايرة، حيث  $k$  قيمة يجب تحديدها.

أ. يَبَيّن أنه يمكن للفرز بالإدراج أن يفرز  $n/k$  لائحة جزئية، طول كل منها  $k$ ، في زمن  $\Theta(nk)$  في أسوأ الحالات.

ب. يَبَيّن كيف يمكن دمج اللوائح الجزئية في زمن  $\Theta(n \lg(n/k))$  في أسوأ الحالات.

ت. إذا علمنا أن زمن تنفيذ الخوارزمية للعدلة في أسوأ الحالات هو  $\Theta(nk + n \lg(n/k))$ ، ما هي أكبر

قيمة ل  $k$  بصفتها دالة ل  $n$  يكون زمن تنفيذ الخوارزمية للعللة عندها مماثلاً لزمن تنفيذ خوارزمية الفرز بالدمج المعيارية، باستخدام تدوين  $\Theta$ ؟

ث. كيف يجب أن نختار  $k$  عملياً؟

## 2-2 صحة الفرز الفقاعي

الفرز الفقاعي bubblesort، خوارزمية فرز شائعة، لكنها غير فعالة، تعمل على التبدل لتكرار للعناصر المتجاورة غير المرتبة.

BUBBLESORT( $A$ )

```

1 for  $i = 1$  to  $A.length - 1$ 
2   for  $j = A.length$  downto  $i + 1$ 
3     if  $A[j] < A[j - 1]$ 
4       exchange  $A[j]$  with  $A[j - 1]$ 
```

أ. لتكن  $A'$  الصيغة التي نشير إلى خرج BUBBLESORT( $A$ ). حتى نثبت أن إجراء BUBBLESORT صحيح، نحتاج إلى إثبات أنه يتوقف، وأن:

$$A'[1] \leq A'[2] \leq \dots \leq A'[n], \quad (3.2)$$

حيث  $n = A.length$ . ما الذي يجب أن نُثبته أيضاً لنبيّن أن BUBBLESORT يُفرّز فعلاً؟

سيثبت الجزءان التاليان صحة المتراجحة (3.2).

ب. أعطِ بدقة لامتغير حلقة for في الأسطر 2-4، وأثبت صحته. يجب أن يستعمل برهانك بنية برهان لامتغير الحلقة الذي عُرض في هذا الفصل.

ث. باستخدام شرط توقف لامتغير الحلقة للبرهان في الجزء (ب)، ضع لامتغير حلقة حلقة for في الأسطر 1-4 يسمح لك برهان المتراجحة (3.2). يجب أن يستعمل برهانك بنية برهان لامتغير الحلقة الذي عُرض في هذا الفصل.

ث. ما هو زمن تنفيذ الفرز الفقاعي في أسوأ الحالات؟ قارنه بزمن تنفيذ الفرز بالإدراج؟

## 3-2 صحة قاعدة هورنر

يُنجزّ مقطع الرماز التالي قاعدة هورنر Horner's rule المستخدمة لتقييم كثير حدود

$$\begin{aligned}
 P(x) &= \sum_{k=0}^n a_k x^k \\
 &= a_0 + x(a_1 + x(a_2 + \dots + x(a_{n-1} + x a_n) \dots)) ,
 \end{aligned}$$

ولنكن لدينا قيم للمعاملات  $a_0, a_1, \dots, a_n$  وقيمة  $x$  معطاة:

```

1  y = 0
2  for i = n downto 0
3      y = ai + x . y

```

أ. ما هو زمن تنفيذ قطعة الرماز السابقة المقابلة لقاعدة هورنر، باستخدام تدوين  $\Theta$  ؟

ب. أكتب شبه رماز لتنجز خوارزمية بسيطة لتقييم كثير حدود تحسب كل حد من كثير الحدود بدءاً من البداية. ما هو زمن تنفيذ هذه الخوارزمية؟ قارنه بزمن تنفيذ قاعدة هورنر؟

ت. ليكن لدينا لامتغير الحلقة التالي:

في بداية كل تكرار الحلقة for التي تشمل السطرين 2-3،

$$y = \sum_{k=0}^{n-(i+1)} a_{k+i+1} x^k .$$

فسر مجموعاً عالياً من الحدود على أنه مساوي 0. باتباع بنية برهان لامتغير الحلقة الذي عُرِضَ في هذا الفصل، استخدم لامتغير الحلقة هذا لإظهار أن  $y = \sum_{k=0}^n a_k x^k$  عند التوقف.

ث. اختتم عملك بمناقشة أن قطعة الرماز المعطى تحسب -على نحو صحيح- كثير حدود موصوفاً بالمعاملات  $a_0, a_1, \dots, a_n$ .

#### 4-2 العكوس

لتكن  $A[1..n]$  صفيفاً مؤلفة من  $n$  عدداً متميزاً. إذا كان  $i < j$  و  $A[i] > A[j]$ ، عندها يسمى الزوج  $(i, j)$  عكساً inversion في  $A$ .

أ. اذكر العكوس الخمسة الموجودة في الصفيفة  $A = (2, 3, 8, 6, 1)$ .

ب. ما الصفيفة التي عناصرها من المجموعة  $\{1, 2, \dots, n\}$  والتي تحتوي أكبر عدد من العكوس؟ وكم عكساً تحتوي؟

ت. ما هي العلاقة بين زمن تنفيذ الفرز بالإدراج وعدد العكوس في صفيفة الدخل؟ علل جوابك.

ث. أعط خوارزمية تحدد عدد العكوس في أي تبديل على  $n$  عنصر، زمن تنفيذها في أسوأ الحالات هو  $\Theta(n \lg n)$ . (نلمح: عُدّل الفرز بالدمج.)

## ملاحظات الفصل

في عام 1968 نشر Knuth الجزء الأول من ثلاثة أجزاء من كتاب بعنوان عام هو فن برمجة الحاسوب *The Art of Computer Programming* [209, 210, 211]. قدّم الجزء الأول للطريقة الحديثة في دراسة خوارزميات الحاسوب التي تركز على تحليل زمن التنفيذ. والسلسلة بأكملها مرجع جذاب وقيم لكثير من المواضيع المعروضة هنا. تُشتق كلمة "خوارزمية" *algorithm*، حسب Knuth، من اسم الخوارزمي، وهو عالم رياضيات فارسي عاش في القرن التاسع.

دافع Aho و Hopcroft و Ullman [5] عن التحليل المقارب للخوارزميات - باستخدام التدوينات التي يرضها الفصل 3، ومنها تدوين  $\Theta$  - بصفتها طريقة لمقارنة الأداء النسبي. وقد أشاع هؤلاء المؤلفون استخدام العلاقات القودية لوصف أزمنة تنفيذ الخوارزميات القودية.

قدّم Knuth [211] معالجة موسوعية لكثير من خوارزميات الفرز. تحتوي مقارنته لخوارزميات الفرز (في الصفحة 381) تحليلات دقيقة تعد الخطوات مشابهة للتحليل الذي قمنا به هنا في حالة الفرز بالإدراج. تتضمن مناقشة Knuth للفرز بالإدراج عدة تعديلات على هذه الخوارزمية. أهمها خوارزمية فرز شيل Shell's sort، التي أدخلها D. L. Shell، والتي تستخدم الفرز بالإدراج على متطلبات جزئية دورية من الدخول لتعطي خوارزمية فرز أسرع.

وصف Knuth أيضًا خوارزمية الفرز بالدمج. وتحديث في كتابه عن آلة دمج ميكانيكية *mechanical collator*، يعود تاريخ اختراعها إلى 1938، كانت قادرة على دمج مجموعتين من البطاقات المثقبة بمرور واحد. ويبدو أن J. Von Neumann أحد الرواد في علوم الحاسوب، كان قد كتب برنامجًا للفرز بالدمج على حاسوب EDVAC عام 1945.

وصف Gries [153] البدايات المبكرة لبرهان صحة البرامج، ونسبها إلى P. Naur صاحب أول مقال في هذا الحقل. ونسب Gries لامتفيزات الحلقة إلى R. W. Floyd. يشرح الكتاب الجامعي لـ Mitchell [256] أحدث التطورات التي طرأت على برهان صحة البرامج.



تمطي مرتبة نمو زمن تنفيذ خوارزمية ما، كما عرّفناها في الفصل الثاني، توصيفًا بسيطًا لفعالية الخوارزمية، ونسمح لنا أيضًا بمقارنة أداء خوارزميات بديلة فيما بينها. فمثلاً، ما إن يصبح حجم المُدخلات  $n$  كبيراً كفاية، حتى يتغلب الفرز بالدمج بزمن تنفيذه في أسوأ الحالات  $\Theta(n \lg n)$  على الفرز بالإدراج ذي زمن التنفيذ في أسوأ الحالات  $\Theta(n^2)$ . ومع أننا نستطيع في بعض الأحيان تحديد زمن التنفيذ الدقيق لخوارزمية ما، كما فعلنا في الفرز بالإدراج في الفصل الثاني، إلا أن الدقة الزائدة لا تستحق عادة العناء المبذول للحصول عليها. ففي حالة مُدخلات كبيرة كفاية، يغطي حجم المُدخلات نفسه على ثوابت الجداء وعلى الحدود من المراتب الصغرى.

عندما ننظر إلى أحجام مُدخلات كبيرة كفاية تكون مرتبة نمو زمن التنفيذ فقط ذات معنى، ونكون بصدد دراسة الفعالية المقاربة *asymptotic* للخوارزميات. أي إننا نختِم بكيفية تزايد زمن تنفيذ خوارزمية ما تبعاً لتزايد حجم المُدخلات عند النهاية، أي عندما يصبح حجم المُدخلات غير محدود. وعادةً، تكون أكثر الخوارزميات فعالية بالمقاربة هي الخيار الأفضل لكل الحجم ما عدا الحجم الصغيرة جدًا.

يقدم هذا الفصل عدّة طرق قياسية لتبسيط التحليل للمقارب للخوارزميات. ويبدأ المقطع التالي بتعريف عدة أنواع من "التدوين للمقارب *asymptotic notation*" التي سبق أن رأينا مثلاً عنها، وهو تدوين  $\Theta$ . ثمّ نقدّم مجموعة من الاصطلاحات التدوينية المستخدمة في هذا الكتاب، وأخيرًا نراجع سلوك الدوال التي نَظهر غالبًا عند تحليل الخوارزميات.

### 1.3 التدوين المقارب

تعتمد التدوينات التي نستخدمها لوصف زمن التنفيذ للمقارب للخوارزمية ما على دوال معرفة على مجموعة الأعداد الطبيعية  $N = \{0, 1, 2, \dots\}$ . وهذه التدوينات مناسبة لوصف دالة زمن التنفيذ في أسوأ الحالات  $T(n)$ ، التي تكون معرفة عادة على أحجام المُدخلات الصحيحة فقط. ومع ذلك، فمن المناسب في بعض الأحيان، أن نقبل ببعض التحايزات في التدوين للمقارب؛ فمثلاً، يمكن بسهولة توسيع التدوين ليشمل حقل

الأعداد الحقيقية، أو على العكس حصره في مجموعة جزئية من الأعداد الطبيعية. إلا أنه من المفهم أن نَمَعِي تمامًا معنى التدوين بحيث لا نسمي استخدامه بوجود هذه التجاوزات. يعرف هذا القطع التدوينات المقاربة الأساسية، ويقدم أيضًا بعض التجاوزات الشائعة.

### التدوين المقارب، والدوال، وأزمنة التنفيذ

سنستخدم التدوين المقارب أساسًا لوصف أزمنة تنفيذ الخوارزميات، مثلما فعلنا عندما ذكرنا أن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو  $\Theta(n^2)$ ، وذلك على الرغم من أن التدوين المقارب يطبق في الواقع على الدوال. نذكر أننا وصفتنا زمن تنفيذ الفرز بالإدراج في أسوأ الحالات بالصيغة  $an^2 + bn + c$ ، حيث  $a$  و  $b$  و  $c$  ثوابت. فإذا كتبنا زمن تنفيذ الفرز بالإدراج بالصيغة  $\Theta(n^2)$ ، نكون قد أغفلنا بعض تفاصيل هذه الدالة. ولما كان التدوين المقارب يطبق على الدوال، فإن ما كتبناه على أنه  $\Theta(n^2)$  هو الدالة  $an^2 + bn + c$  التي تصادف أنها توصف زمن تنفيذ الفرز بالإدراج في أسوأ الحالات.

في هذا الكتاب، ستكون الدوال التي سنطبق عليها التدوين المقارب في غالب الأحيان توصيفات لأزمنة تنفيذ خوارزميات. إلا أن التدوين المقارب يمكن أن ينطبق على دوال توصف جوانب أخرى من الخوارزميات (مثلًا، حجم الذاكرة الذي تشغله)، أو حتى على دوال لا علاقة لها بالبنية بالخوارزميات.

وحتى عندما نستخدم التدوين المقارب لتطبيقه على زمن تنفيذ خوارزمية ما، نحتاج إلى فهم أي زمن تنفيذ نَمَعِي. ففي بعض الأحيان، نَحْمِ بزمن التنفيذ في أسوأ الحالات، إلا أننا كثيرًا ما نرغب في توصيف زمن التنفيذ مهما كان الدخل. وبعبارة أخرى، كثيرًا ما نرغب في إعطاء تصريح يشمل المدخلات كلها، وليس مدخلات أسوأ الحالات فحسب. وسنطالع على تدوينات مقاربة مناسبة لتوصيف أزمنة التنفيذ مهما كان الدخل.

### تدوين $\Theta$

وجدنا في الفصل الثاني أن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو  $\Theta(n^2)$ . سنعرّف مفهوم هذا التدوين لدالة معطاة  $g(n)$ ، ونرمز له بـ  $\Theta(g(n))$ ، بأنه مجموعة الدوال:

$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2 \text{ and } n_0 \text{ such that}$

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ for all } n \geq n_0\}^1$$

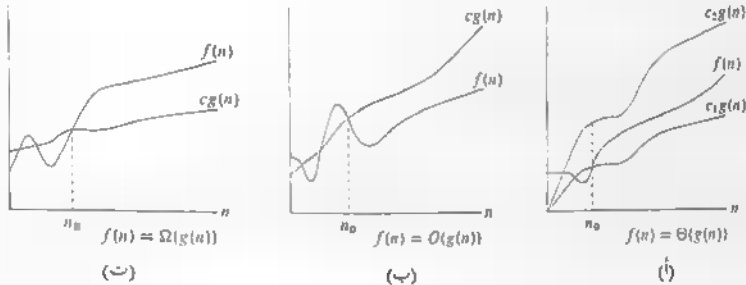
تنتمي دالة  $f(n)$  إلى المجموعة  $\Theta(g(n))$  إذا كان هناك ثابتان  $c_1$  و  $c_2$  بحيث يمكن "حشر sandwich" هذه الدالة بين  $c_1 g(n)$  و  $c_2 g(n)$ ، عندما تكون  $n$  كبيرة كفاية. ولما كانت  $\Theta(g(n))$  مجموعة، كان

<sup>1</sup> النقطتان "؛" تُقرأان في تدوين المجموعات "حيث".

يمكننا أن نكتب " $f(n) \in \Theta(g(n))$ " لنتبين أن  $f(n)$  هي عنصر من  $\Theta(g(n))$ . ولكننا بدلاً من ذلك نكتب عادة " $f(n) = \Theta(g(n))$ " للتعبير عن المفهوم نفسه. قد يبدو هذا التجاوز باستخدام المساواة عوضاً عن إشارة الانتماء في البداية مشوشاً بعض الشيء، ولكننا سنرى بعد قليل في هذا القطع أن له فوائده.

يعطي الشكل 1.3 (أ) عقيداً مبسطاً للدوال  $f(n)$  و  $g(n)$ ، حيث  $f(n) = \Theta(g(n))$ . تقع قيمة  $f(n)$  فوق  $c_1 g(n)$  وأسفل  $c_2 g(n)$ ، أيًا كانت قيمة  $n$  الواقعة إلى يمين  $n_0$ . وبعبارة أخرى، أيًا كانت  $n \geq n_0$ ، فإن الدالة  $f(n)$  تساوي  $g(n)$  ضمن مُعامل ثابت. ونقول إن الدالة  $g(n)$  هي حدّه مُخفّفهم بالمقاربة *asymptotically tight bound* للدالة  $f(n)$ .

يتطلب تعريف  $\Theta(g(n))$  أن يكون كل عنصر  $f(n) \in \Theta(g(n))$  موجّباً بالمقاربة *asymptotically nonnegative*، أي أن تكون الدالة  $f(n)$  موجبة عندما تصبح  $n$  كبيرة كفاية. والدالة الموجبة تماماً بالمقاربة *asymptotically positive* هي دالة موجبة تماماً لكل قيم  $n$  الكبيرة كفاية. بالنتيجة، يجب أن تكون الدالة  $g(n)$  نفسها موجبة بالمقاربة، وإلا فإن المجموعة  $\Theta(g(n))$  تكون فارغة. ولذلك نفترض أن كل الدوال المستخدمة في تدوين  $\Theta$  هي موجبة بالمقاربة. وتبقى هذه الفرضية محققة أيضاً في بقية التدوينات المقارنة التي ستعرّفها في هذا الفصل.



الشكل 1.3 أمثلة بيانية للتدوينات  $\Theta$ ،  $O$ ، و  $\Omega$ . إن قيمة  $n_0$  المبينة في كل جزء من الشكل هي أصغر قيمة ممكنة، ويمكن أن نحل محلها أية قيمة أكبر منها. (أ) يُحدّد تدوين  $\Theta$  دالة ما ضمن معاملين ثابتين. ونكتب  $f(n) = \Theta(g(n))$  إذا وُجدت ثوابت موجبة  $n_0$ ، و  $c_1$  و  $c_2$  بحيث، عندما تقع  $n$  عند  $n_0$  وإلى يمينها، تقع قيمة  $f(n)$  دائماً بين  $c_1 g(n)$  و  $c_2 g(n)$  أو تساويهما. (ب) يعطي تدوين  $O$  حدّاً أعلى لدالة ما ضمن معامل ثابت. ونكتب  $f(n) = O(g(n))$  إذا وُجد ثابتان موجبان  $n_0$  و  $c$  بحيث، عندما تقع  $n$  عند  $n_0$  وإلى يمينها، تقع قيمة  $f(n)$  دائماً على  $c g(n)$  أو تحته. (ت) يعطي تدوين  $\Omega$  حدّاً أدنى لدالة ما ضمن معامل ثابت. ونكتب  $f(n) = \Omega(g(n))$  إذا وُجد ثابتان موجبان  $n_0$  و  $c$  بحيث، عندما تقع  $n$  عند  $n_0$  وإلى يمينها، تقع قيمة  $f(n)$  دائماً على  $c g(n)$  أو فوقه.

فَدَمْنَا في الفصل الثاني تفسيرًا تقريبيًا لمفهوم تدوين- $\Theta$  يتمثل في التخلص من الحدود الأدنى مرتبة وفي تجاهل المعامل المرافق للحد الأعلى مرتبة. دعنا نبرز بإيجاز هذا الحُلس باستخدام التعريف الصوري لنبين أن  $\Theta(n^2) = 3n - \frac{1}{2}n^2$ ، وللقيام بذلك، يجب أن نحدد الثوابت الموجبة  $c_1$  و  $c_2$  و  $n_0$  بحيث يكون

$$c_1 n^2 \leq \frac{1}{2} n^2 - 3n \leq c_2 n^2$$

أيًا كانت  $n_0 \geq n$ . وبالتقسيم على  $n^2$  نحصل على

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2.$$

يمكن تحقيق المتراجحة اليمنى أيًا كانت  $n \geq 1$  باختيار  $c_2 \geq 1/2$ . وبالمثل يمكن تحقيق المتراجحة اليسرى أيًا كانت  $n \geq 7$  باختيار  $c_1 \leq 1/14$ . وهكذا، وباختيار  $c_1 = 1/14$  و  $c_2 = 1/2$  و  $n_0 = 7$ ، يمكن أن نتحقق من أن  $\Theta(n^2) = 3n - \frac{1}{2}n^2$ . طبعًا يوجد خيارات أخرى لقيم الثوابت، ولكن ما يهم هنا هو وجود قيم ممكنة. لاحظ أن هذه الثوابت تتعلق بالدالة  $\frac{1}{2}n^2 - 3n$ ؛ وأية دالة أخرى من  $\Theta(n^2)$  تتطلب عادةً ثوابت مختلفة.

بوسعنا أيضًا استخدام التعريف الصوري للتحقق من أن  $\Theta(n^2) \neq 6n^3$ . افترض، بهدف الوصول إلى تناقض، أنه يوجد  $c_2$  و  $n_0$  بحيث يكون  $6n^3 \leq c_2 n^2$  أيًا كانت  $n_0 \geq n$ . ولكن بالتقسيم على  $n^2$  نحصل على  $c_2/6 \leq n$  والذي لا يمكن أن يتحقق عندما تكون  $n$  كبيرة كفاية، وذلك لأن  $c_2$  ثابت.

إذن، يمكن بالحُلس تجاهل الحدود الأدنى مرتبة في دالة موجبة بالمقارنة عند تحديد حدود محكمة بالمقارنة، لأنها تكون غير ذات معنى في حالة قيم كبيرة لـ  $n$ . إن جزءًا صغيرًا من الحد الأعلى مرتبة كافٍ ليقوق هذه الحدود الأدنى مرتبة. وهكذا، يسمح إعطاء قيمة لـ  $c_1$  أصغر قليلًا من معامل الحد الأعلى مرتبة، وإعطاء قيمة لـ  $c_2$  أكبر قليلًا من هذا المعامل، بتحقيق المتراجحات في تعريف التدوين- $\Theta$ . ويمكن تجاهل معامل الحد الأعلى مرتبة أيضًا، لأنه يغير فقط قيم  $c_1$  و  $c_2$  بمعامل ثابت مساوٍ لهذا المعامل.

كمثال على ذلك، لنأخذ أية دالة تربيعية  $f(n) = an^2 + bn + c$ ، حيث  $a$  و  $b$  و  $c$  ثوابت و  $a > 0$ . إن إهمال الحدود الأدنى مرتبة وتجاهل الثابت يعطينا  $f(n) = \Theta(n^2)$ . حتى نتوصل للنتيجة نفسها بطريقة صورية، نأخذ  $c_1 = a/4$  و  $c_2 = 7a/4$  و  $n_0 = 2 \cdot \max(|b|/a, \sqrt{|c|/a})$ . يمكن للقارئ أن يتحقق من أن  $0 \leq c_1 n^2 \leq an^2 + bn + c \leq c_2 n^2$  أيًا كانت  $n_0 \geq n$ . وعمومًا في حالة أي كثير حدود  $p(n) = \sum_{i=0}^d a_i n^i$ ، حيث  $a_d > 0$  و  $a_i$  ثوابت، يكون لدينا  $p(n) = \Theta(n^d)$  (انظر للسألة (١-3)).

ولما كان أي ثابت هو كثير حدود من الدرجة 0، فيمكن أن نعبّر عن أية دالة ثابتة بأنها  $\Theta(n^0)$ ،

أو  $\Theta(1)$ . إن التدوين الثاني هو تجاوز بسيط، إذ لا يظهر فيه بوضوح للتحويل الذي يسمى إلى اللانهاية.<sup>2</sup> مستخدم التدوين  $\Theta(1)$  مراراً يعني به إما ثابتاً أو دالة ثابتة بالنسبة للتحويل ما.

### تدوين- $O$

إن التدوين- $\Theta$  يحدّ دالة ما بالمقارنة من الأعلى ومن الأسفل. عندما يكون لدينا حد أعلى بالمقارنة  $asymptotic upper bound$  فقط، فإننا نستخدم التدوين- $O$ . فإذا كانت  $g(n)$  دالة معطاة، فإننا نعي بالمعارة  $O(g(n))$  (والتي تلفظ "big-oh of  $g$  of  $n$ "، أو أحياناً "oh of  $g$  of  $n$ " فقط) مجموعة الدوال

$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$

$$0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}.$$

نستخدم تدوين- $O$  لنعطي حدّاً أعلى للدالة ما مضروباً بثابت، يبيّن الشكل 1.3 (ب) المجلس الذي يرتكز عليه تدوين- $O$ : لكل قيم  $n$  عند  $n_0$  وإلى يمينها، تكون قيمة الدالة  $f(n)$  مساوية لـ  $cg(n)$  أو أقل منها.

نكتب  $f(n) = O(g(n))$  لنشير إلى أن الدالة  $f(n)$  هي عنصر من المجموعة  $O(g(n))$ . لاحظ أنّ  $f(n) = \Theta(g(n))$  يقتضي أن  $f(n) = O(g(n))$ ، لأنّ تدوين- $\Theta$  أقوى من تدوين- $O$ . وباستخدام لغة نظرية المجموعات، يمكننا كتابة  $\Theta(g(n)) \sqsubseteq O(g(n))$ . إذن، إن برهاننا بأنّ دالة تربيعية  $an^2 + bn + c$  حيث  $a > 0$ ، هي  $\Theta(n^2)$ ، يبيّن أيضاً أنّ دالة تربيعية تنتمي إلى  $O(n^2)$ . وقد يكون مفيداً أكثر أنّ دالة  $an + b$  تعطى عندما  $a > 0$ ، تنتمي إلى  $O(n^2)$ ، وهذا ما يمكن التحقق منه بسهولة بأخذ  $c = a + |b|$  و  $n_0 = \max(1, -b/a)$ .

قد يجد بعض القراء الذين تعرّضوا لتدوين- $O$  من قبل غربة في أن نكتب مثلاً  $n = O(n^2)$ . يُستخدم تدوين- $O$  في الأدبيات أحياناً لوصف حدود محكمة بالمقارنة، وهذا ما عرفناه باستخدام تدوين- $\Theta$ . إلا أنه، عندما نكتب في هذا الكتاب  $f(n) = O(g(n))$  فإننا ندّعي فقط أن هناك مضاعفاً ثابتاً لـ  $g(n)$  هو حد أعلى بالمقارنة لـ  $f(n)$ ، دون أيّة ادعاءات عن مدى إحكام هذا الحد الأعلى. إن التمييز بين الحدود العليا بالمقارنة والحدود المحكمة بالمقارنة أصبح الآن متعارفاً في أدبيات الخوارزميات.

يمكننا في كثير من الأحيان، باستخدام تدوين- $O$ ، وصف زمن تنفيذ إجرائية ما بتفحص البنية العامة للخوارزمية فقط. فعلى سبيل المثال، تعطي بنية الحلقة للتداخلتين في خوارزمية الفرز بالإدراج - التي تعرّضنا لها في الفصل الثاني - مباشرة حدّاً أعلى لزمن التنفيذ في أسوأ الحالات هو  $O(n^2)$ ، فكلّفة كل تكرار للحلقة

<sup>2</sup> للمشكلة الحقيقية هي أن التدوين المعتاد للدوال لا يميّز بين الدوال والقيم؛ ففي حساب- $\lambda$ -calculus، تكون متوسطات الدالة عمدة بوضوح؛ إذ يمكن كتابة الدالة  $n^2$  بالصيغة  $\lambda n. n^2$  أو حتى بالصيغة  $\lambda x. x^2$ . إلا أنّ اعتماد طريقة تدوين أكثر دقة سيعقد المعالجة الجبرية، ولذلك فقد اخترنا أن نسمح بهذا التجاوز.

الداخلية محدود من الأعلى بـ  $O(1)$  (أي ثابت)، وأعلى قيمة لكل من الدليلين  $i$  و  $j$  هي  $n$  على الأكثر، ونُتخذ الحلقة الداخلية مرة واحدة على الأكثر لكل  $n^2$  زوجًا من قيم  $i$  و  $j$ .

لما كان تدوين- $O$  يصف حدًا أعلى، فنعلمنا نستخدمه لإعطاء حدّ لزمان تنفيذ خوارزمية ما في أسوأ الحالات، فنحصل على حد لزمان تنفيذ الخوارزمية على أي مُدخل. وهكذا، فإن الحدّ  $O(n^2)$  لزمان تنفيذ الفرز بالإدراج في أسوأ الحالات ينطبق أيضًا على زمن تنفيذ هذه الخوارزمية على أي مُدخل. أما فيما يخص الحدّ  $\Theta(n^2)$  لزمان تنفيذ الفرز بالإدراج في أسوأ الحالات، فإن ذلك لا يقتضي أن هناك حدًا  $\Theta(n^2)$  على زمن تنفيذ الفرز بالإدراج في أسوأ الحالات على أي مُدخل. فعلى سبيل المثال، رأينا في الفصل الثاني، أنه عندما يكون المُدخل مفرورًا سلفًا، فإن الفرز بالإدراج يُنقذ في زمن  $\Theta(n)$ .

رياضيًا، يمثل قولنا إن زمن تنفيذ الفرز بالإدراج هو  $O(n^2)$  محاولة، وذلك لأن زمن التنفيذ الفعلي عندما  $n$  محدّدة يتغيّر بتغيّر المُدخل ذي الحجم  $n$ . ولكن ما نعبه عندما نقول "زمن التنفيذ هو  $O(n^2)$ " هو أن هناك دالة  $f(n)$  تحقّق أنّها  $O(n^2)$  بحيث أنه أيّا كانت قيمة  $n$ ، وأيّا كان المُدخل ذو الحجم  $n$  الذي نختاره، فإن زمن التنفيذ لهذا المُدخل محدود من الأعلى بقيمة  $f(n)$ . وهذا ما يكافئ قولنا إن زمن التنفيذ في أسوأ الحالات هو  $O(n^2)$ .

### تدوين- $\Omega$

يقدم تدوين- $\Omega$  حدًا أدنى بالمقاربة *asymptotic lower bound*، تمامًا مثلما يقدم تدوين- $O$  حدًا أعلى بالمقاربة. فإذا كانت  $g(n)$  دالة معطاة، فإننا نعبى بالمعبرة  $\Omega(g(n))$  [والتي تلفظ "big-omega of  $g$  of  $n$ "]، أو أحيانًا "omega of  $g$  of  $n$ " فقط] مجموعة الدوال

$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$

$$0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}.$$

يبين الشكل 1.3 (ت) الحدس الذي يركّز عليه تدوين- $\Omega$ : لكل قيم  $n$  عند  $n_0$  وإلى بعدها، تكون قيمة الدالة  $f(n)$  مساوية لـ  $cg(n)$  أو أعلى منها.

اعتمادًا على تعاريف التدوينات للمقاربة التي رأيناها حتى الآن، من السهل أن نبرهن للمبرهنة الهامة التالية (انظر التمرين 5-1.3).

### مبرهنة 1.3

أيّا كانت الدالتان  $f(n)$  و  $g(n)$ ، يكون لدينا  $f(n) = \Theta(g(n))$  إذا وفقط إذا كانت  $f(n) = O(g(n))$  و  $f(n) = \Omega(g(n))$ .

كمثال على تطبيق هذه البرهنة، برهاننا بأن  $an^2 + bn + c = \Theta(n^2)$  أيًا كانت الثوابت  $a$  و  $b$  و  $c$ ، حيث  $a > 0$ ، الذي نستنتج منه مباشرة أن  $an^2 + bn + c = \Omega(n^2)$  و  $an^2 + bn + c = O(n^2)$ . عمليًا، بدلاً من استخدام البرهنة 1.3 للحصول على حدين أعلى وأدنى من حدود محكمة بالمقارنة - كما فعلنا في هذا المثال - فإننا نستخدمها عادةً للبرهان على حدود محكمة بالمقارنة بدءًا من حدود عليا ودنيا بالمقارنة.

عندما نقول إن زمن تنفيذ أية خوارزمية (دون تحديد إضافي) هو  $\Omega(g(n))$ ، فإننا نعني أنه مهما كان المدخل المحدد ذو الحجم  $n$  الذي نختاره لكل قيم  $n$ ، فإن زمن التنفيذ على هذا المدخل، عندما تكون  $n$  كبيرة كفاية، هو على الأقل مضاعف ثابت من  $g(n)$ . وهذا يكافئ قولنا إننا نعطي حدًا أدنى لزمن تنفيذ خوارزمية في أحسن الحالات. فمثلًا، زمن تنفيذ الفرز بالإدراج في أحسن الحالات هو  $\Omega(n)$ ، وهذا يقتضي أن زمن تنفيذ الفرز بالإدراج هو  $\Omega(n)$ .

إذن، يشتمل زمن تنفيذ الفرز بالإدراج إلى  $\Omega(n)$  و  $O(n^2)$ ، إذ إنه يقع في أي مكان بين دالة خطية ما ل  $n$  ودالة تربيعية ل  $n$ . إضافة إلى ذلك، فإن هذه الحدود تكون محكمة بالمقارنة قدر الإمكان: فمثلًا، زمن تنفيذ الفرز بالإدراج ليس  $\Omega(n^2)$ ، لأنه يوجد مُدْخَل يُنفِّذ الفرز بالإدراج عليه في زمن  $\Theta(n)$  (مثال ذلك، إذا كان المدخل مفرورًا سلفًا). ومع ذلك، فهذا لا يناقض قولنا إن زمن تنفيذ الفرز بالإدراج في أسوأ الحالات هو  $\Omega(n^2)$ ، وذلك لأنه يوجد مُدْخَل يجعل الخوارزمية تستغرق زمنًا  $\Omega(n^2)$ .

### التدوين المقارب في المعادلات والمترجمات

رأينا سابقًا كيف يمكن استخدام التدوين المقارب داخل الصيغ الرياضية. مثالًا: عندما قدّمنا للتدوين  $O$ ، كتبنا: " $n = O(n^2)$ ". وقد نكتب أيضًا  $2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ . ولكن كيف نفسر مثل هذه الصيغ؟

عندما يكون التدوين المقارب وحيًا (أي إنه لا يمثل جزءًا من صيغة أكبر منه) على الطرف الأيمن للمعادلة (أو للمترجمة)، كما في  $n = O(n^2)$ ، فقد عرّفنا سابقًا إشارة المساواة على أنها تعني الانتماء إلى المجموعات:  $n \in O(n^2)$ . ومع ذلك، إذا ظهر التدوين المقارب في صيغة ما، فإننا نفسره عمومًا، على أنه يحل محل دالة غير مسماة لا يهمنا كثيرًا أن نعطيها اسمًا؛ فمثلًا، تعني الصيغة  $\Theta(n)$  وفي هذه الحالة نجعل  $f(n) = 3n + 1$ ، حيث  $2n^2 + 3n + 1 = 2n^2 + f(n)$ ، أي فعلًا من  $\Theta(n)$ .

يمكن أن يساعد استخدام التدوين المقارب بهذه الطريقة على حذف التفاصيل غير الضرورية في معادلة ما. مثالًا، عرّفنا في الفصل الثاني عن زمن تنفيذ الفرز بالدمج في أسوأ الحالات باستخدام العلاقة العودية

$$T(n) = 2T(n/2) + \Theta(n) .$$

فإذا كنا معنيين فقط بالسلوك المقارب لـ  $T(n)$ ، فلا داعي لتحديد كل الحدود من الدرجة الأصغر بدقة؛ ومن المفهوم أنها تدخل كلها في الدالة غير المسماة التي يشار إليها بالحد  $\Theta(n)$ .

نشير أيضاً إلى أننا نفهم أن عدد الدوال غير المسماة في عبارة ما يكون مساوياً لعدد لمرات التي يظهر فيها التدوين المقارب. فمثلاً، في العبارة

$$\sum_{i=1}^n O(i),$$

يوجد فقط دالة غير معروفة وحيدة (هي دالة  $i$ ). وهكذا، فإن هذه العبارة تختلف عن العبارة:

$$O(1) + O(2) + \dots + O(n)$$

يظهر التدوين المقارب في بعض الأحيان في الطرف الأيسر من معادلة كما في

$$2n^2 + \Theta(n) = \Theta(n^2).$$

نفسر مثل هذه المعادلات باستخدام القاعدة التالية؛ مهما كانت طريقة اختيار الدوال غير المسماة إلى يسار إشارة مساواة، هناك طريقة لاختيار الدوال غير المسماة إلى يمين إشارة المساواة بحيث تكون المعادلة محققة.

إذن، يكون معنى المعادلة في مثالنا السابق هو أنه في حالة أية دالة  $f(n) \in \Theta(n)$ ، توجد دالة ما  $g(n) \in \Theta(n^2)$  بحيث يكون  $2n^2 + f(n) = g(n)$  لكل قيم  $n$ . وبعبارة أخرى، يقدم الطرف الأيمن للمعادلة مستوى أقل تفصيلاً من الطرف الأيسر.

ويمكن سُلْطَنَة عدّة علاقات من هذا النوع مفاكماً في

$$\begin{aligned} 2n^2 + 3n + 1 &= 2n^2 + \Theta(n) \\ &= \Theta(n^2). \end{aligned}$$

ويمكن تفسير كل معادلة على حدة باستخدام القاعدة السابقة. فالمعادلة الأولى تعني أنه توجد دالة  $f(n) \in \Theta(n)$  بحيث يكون  $2n^2 + 3n + 1 = 2n^2 + f(n)$  لكل قيم  $n$ . وتعني المعادلة الثانية أنه في حالة أية دالة  $g(n) \in \Theta(n^2)$  (كالدالة  $f(n)$  التي ورد ذكرها الآن) توجد دالة  $h(n) \in \Theta(n^2)$  بحيث يكون  $2n^2 + g(n) = h(n)$  لكل قيم  $n$ . لاحظ أن هذا التفسير يؤدي إلى  $2n^2 + 3n + 1 = \Theta(n^2)$ ، وهذا ما تبينه لنا ببساطة سُلْطَنَة للمعادلات.

### تدوين $\Theta$ -

إن الحد الأعلى للمقارب الذي يقدمه تدوين  $\Theta$  قد يكون مُحْكَمًا بالمقاربة، وقد لا يكون كذلك. فمثلاً الحد  $2n^2 = O(n^2)$  مُحْكَمٌ بالمقاربة، غير أن الحد  $2n = O(n^2)$  ليس كذلك. نستخدم تدوين  $\Theta$  لنعرّف عن حد أعلى غير محكم بالمقاربة. نعرّف  $o(g(n))$  صوريّاً [وتلفظ "little-oh of  $g$  of  $n$ "] على أنها المجموعة:



$o(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant}$

$$0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}.$$

مثلاً  $2n^2 \neq o(n^2)$ ، فيما  $2n = o(n^2)$ .

إن تعريفي تدوين- $O$  وتدوين- $o$  متشابهان؛ والفرق الأساسي بينهما هو أنه عندما  $f(n) = O(g(n))$  فإن المتراجحة  $0 \leq f(n) \leq cg(n)$  تكون محققة في حالة ثابت ما  $c > 0$ ، أما في حالة  $f(n) = o(g(n))$ ، فإن للمتراجحة  $0 \leq f(n) < cg(n)$  تكون محققة أياً كان الثابت  $c > 0$ . يمكننا ببساطة أن نقول إنه في التدوين- $o$  تصبح الدالة  $f(n)$  مهملة بالنسبة إلى  $g(n)$  عندما تسمى  $n$  إلى اللانهاية؛ أي:

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0. \quad (1.3)$$

وستخدم بعض المؤلفين هذه النهاية كتعريف للتدوين- $o$ ؛ ونحن في هذا الكتاب نشترط أن تكون الدوال غير المسماة موجبة بالمقارنة.

تدوين- $\omega$

بالمثل، نمثل علاقة تدوين- $\omega$  بتدوين- $\Omega$  علاقة تدوين- $o$  بتدوين- $O$ . نستخدم تدوين- $\omega$  لنشير إلى حد أدنى غير محكم بالمقارنة. وإحدى طرائق تعريف ذلك هي:

$$\omega(g(n)) \in f(n) \text{ إذا وقط إذا } g(n) \in o(f(n)).$$

ومع ذلك، فإننا نعرف  $\omega(g(n))$  صورياً [وتلفظ "little-omega of  $g$  of  $n$ "] على أنها المجموعة:

$$\omega(g(n)) = \{f(n) : \text{for any positive constant } c > 0, \text{ there exists a constant}$$

$$n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}.$$

فمثلاً  $n^2/2 = \omega(n^2)$ ، فيما  $n^2/2 \neq \omega(n^2)$ ، ونقتضي العلاقة  $f(n) = \omega(g(n))$  أن يكون

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty,$$

إذا كانت هذه النهاية موجودة فعلاً. أي إن قيم  $f(n)$  تصبح كبيرة بلا حدود مقارنة بـ  $g(n)$  عندما تسمى  $n$  إلى اللانهاية.

مقارنة الدوال

تنطبق العديد من الخصائص العلائقية للأعداد الحقيقية على المقارنات بالمقارنة. نفترض فيما يلي أن  $f(n)$  و  $g(n)$  موجبان بالمقارنة.

الصعدي:

$$f(n) = \Theta(g(n)) \text{ و } g(n) = \Theta(h(n)) \text{ يقتضيان } f(n) = \Theta(h(n))$$

$$f(n) = O(g(n)) \text{ و } g(n) = O(h(n)) \text{ يقتضيان } f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ و } g(n) = \Omega(h(n)) \text{ يقتضيان } f(n) = \Omega(h(n))$$

$$f(n) = o(g(n)) \text{ و } g(n) = o(h(n)) \text{ يقتضيان } f(n) = o(h(n))$$

$$f(n) = \omega(g(n)) \text{ و } g(n) = \omega(h(n)) \text{ يقتضيان } f(n) = \omega(h(n))$$

الانعكاسية:

$$f(n) = \Theta(f(n))$$

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

التناظرية:

$$f(n) = \Theta(g(n)) \text{ إذا وفقط إذا } g(n) = \Theta(f(n))$$

التناظرية المنقولة:

$$f(n) = O(g(n)) \text{ إذا وفقط إذا } g(n) = \Omega(f(n))$$

$$f(n) = o(g(n)) \text{ إذا وفقط إذا } g(n) = \omega(f(n))$$

ولما كانت هذه الخصائص محققة في التدوينات المقارنة، فيمكننا أن نبني مقابلةً بين المقارنة بالمقارنة بين

دالتين  $f$  و  $g$ ، والمقارنة بين عددين حقيقيين  $a$  و  $b$ :

$$f(n) = O(g(n)) \text{ مثل } a \leq b$$

$$f(n) = \Omega(g(n)) \text{ مثل } a \geq b$$

$$f(n) = \Theta(g(n)) \text{ مثل } a = b$$

$$f(n) = o(g(n)) \text{ مثل } a < b$$

$$f(n) = \omega(g(n)) \text{ مثل } a > b$$

نقول إن الدالة  $f(n)$  أصغر بالمقارنة *asymptotically smaller* من  $g(n)$  إذا كان  $f(n) = o(g(n))$

وإن  $f(n)$  هي أكبر بالمقارنة *asymptotically larger* من  $g(n)$  إذا كان  $f(n) = \omega(g(n))$ .

إلا أن هناك خاصية من خواص الأعداد الحقيقية لا تنطبق على التدوين للمقارب وهي:

الفصل الثلاثي: في حالة أي عددين حقيقيين  $a$  و  $b$ ، يجب أن تتحقق حكمًا إحدى الحالات الثلاث

التالية:  $a < b$  أو  $a = b$  أو  $a > b$ .

ومع أنه يمكن إجراء مقارنة بين أيّ عددين حقيقيّين؛ فلا يمكن إجراء مقارنة بالمقاربة بين أيّ دالتين. أي إنه إذا كانت لدينا دالتان  $f(n)$  و  $g(n)$ ، فيمكن ألاّ يتحقق  $f(n) = O(g(n))$  ولا يتحقق  $f(n) = \Omega(g(n))$ . فمثلاً لا يمكن مقارنة الدالتين  $n$  و  $n^{1+\sin n}$  باستخدام التدوين المقارب، لأن قيمة الأس في  $n^{1+\sin n}$  تتذبذب بين  $\frac{1}{2}$  و  $2$ ، وتأخذ كل القيم فيما بينهما.

### تعاريف

#### 1-1.3

لتكن  $f(n)$  و  $g(n)$  دالتين موجبتين بالمقاربة. استخدم التعريف الأساسي لتدوين- $\Theta$  كي تهرن على أن

$$\max(f(n), g(n)) = \Theta(f(n) + g(n)).$$

#### 2-1.3

برهن أنه في حالة أي ثابتين حقيقيّين  $a$  و  $b$  حيث  $b > \frac{1}{a}$ ، يكون

$$(n + a)^b = \Theta(n^b). \quad (2.3)$$

#### 3-1.3

اشرح سبب كون العبارة "زمن تنفيذ الخوارزمية  $A$  هو على الأقل  $O(n^2)$ " لا معنى لها.

#### 4-1.3

هل  $2^{n+1} = O(2^n)$  وهل  $2^{2n} = O(2^n)$ ؟

#### 5-1.3

أثبت صحة المبرهنة 1.3.

#### 6-1.3

برهن أن زمن تنفيذ خوارزمية ما هو  $\Theta(g(n))$  إذا وفقط إذا كان زمن تنفيذها في أسوأ الحالات هو  $O(g(n))$ ، وزمن تنفيذها في أحسن الحالات هو  $\Omega(g(n))$ .

#### 7-1.3

برهن أن  $\omega(g(n)) \cap o(g(n))$  هو المجموعة الخالية.

#### 8-1.3

يمكن أن نعتمد التدوين الذي نعتمد له حالة متوسطين  $n$  و  $m$  يمكن أن يسعيا إلى اللانهاية على نحو مستقل بمعدلين مختلفين. في حالة دالة معطاة  $g(n, m)$ ، نعي بالتدوين  $O(g(n, m))$  مجموعة الدوال

$$O(g(n, m)) = \{f(n, m) : \text{there exist positive constants } c, n_0, \text{ and } m_0 \\ \text{such that } \frac{1}{c} \leq f(n, m) \leq c g(n, m) \\ \text{for all } n \geq n_0 \text{ or } m \geq m_0\}.$$

أعط التعاريف المقابلة لـ  $\Omega(g(n, m))$  و  $\Theta(g(n, m))$ .

## 2.3 تدوينات قياسية ودوال شائعة

يراجع هذا المقطع بعض الدوال الرياضية والتدوينات القياسية، ويدرس العلاقات فيما بينها. ويوضح أيضاً استخدامات التدوينات المقارنة.

### الاطراد

نقول عن دالة  $f(n)$  إنها **متزايدة باطراد** *monotonically increasing* إذا كان  $m \leq n$  يقتضي أن يكون  $f(m) \leq f(n)$ . وبالمثل، تكون الدالة **متناقصة باطراد** *monotonically decreasing* إذا كان  $m \leq n$  يقتضي  $f(m) \geq f(n)$ . وتكون الدالة  $f(n)$  **متزايدة تماماً** *strictly increasing* إذا كان  $m < n$  يقتضي أن يكون  $f(m) < f(n)$ ، و**متناقصة تماماً** *strictly decreasing* إذا كان  $m < n$  يقتضي  $f(m) > f(n)$ .

### الأرضيات والسقوف

إذا كان  $x$  عدداً حقيقياً، فإننا نرمز إلى أكبر عدد صحيح أصغر أو يساوي  $x$  بالرمز  $[x]$  (نقرأ "أرضية  $x$ "، وإلى أصغر عدد صحيح أكبر أو يساوي  $x$  بالرمز  $\lceil x \rceil$  (نقرأ "سقف  $x$ ". ويكون:

$$x - 1 < [x] \leq x \leq \lceil x \rceil < x + 1. \quad (3.3)$$

أيما كان العدد الحقيقي.  
ويكون:

$$[n/2] + \lceil n/2 \rceil = n,$$

أيما كان العدد الصحيح  $n$ .  
ويكون:

$$\left\lceil \frac{[x/a]}{b} \right\rceil = \left\lceil \frac{x}{ab} \right\rceil, \quad (4.3)$$

$$\left\lfloor \frac{\lceil x/a \rceil}{b} \right\rfloor = \left\lfloor \frac{x}{ab} \right\rfloor, \quad (5.3)$$

$$\left\lfloor \frac{a}{b} \right\rfloor \leq \frac{a + (b - 1)}{b}, \quad (6.3)$$

$$\left\lceil \frac{a}{b} \right\rceil \geq \frac{a - (b - 1)}{b}. \quad (7.3)$$

أيما كان العدد الحقيقي  $x \geq 0$  والعددين الصحيحان  $a, b > 0$ .

إن دالة الأرضية  $[x] = f(x)$  متزايدة باطراد، كما هو الحال في دالة السقف  $\lceil x \rceil = f(x)$ .

### الحساب بالمقاس

إذا كان  $a$  عددًا صحيحًا و  $n$  عددًا صحيحًا موجبًا، فإن  $a \bmod n$  هي باقي قسمة (*remainder*) أو  $a/n$  (*residue*)، أي:

$$a \bmod n = a - n[a/n] . \quad (8.3)$$

ويستج عن ذلك:

$$0 \leq a \bmod n < n . \quad (9.3)$$

من المناسب بعد إعطاء تعريف جيد لمفهوم باقي قسمة عدد صحيح على آخر، أن يكون هناك تدوينًا خاصًا لبيان تساوي باقيي القسمة. فإذا كان  $(a \bmod n) = (b \bmod n)$ ، فإننا نكتب  $a \equiv b \pmod{n}$ ، ونقول إن  $a$  **يكافئ**  $b$  (*equivalent to*) بالمقاس  $n$ . وبعبارة أخرى: إن  $a \equiv b \pmod{n}$  إذا تساوى باقيا قسمة كل من  $a$  و  $b$  على  $n$ . وهذا يكافئ قولنا إن  $a \equiv b \pmod{n}$  إذا وفقط إذا كان  $n$  يقسم  $b - a$ . ونكتب  $a \not\equiv b \pmod{n}$  إذا كانت  $a$  لا تكافئ  $b$  بالمقاس  $n$ .

### كثيرات الحدود

ليكن  $\equiv$  عددًا طبيعيًا، إن كثير حدود في  $n$  من الدرجة  $d$  هو دالة  $p(n)$  صيغتها:

$$p(n) = \sum_{i=0}^d a_i n^i$$

حيث تمثل الثوابت  $a_0, a_1, \dots, a_d$  معاملات *coefficients* كثير الحدود، و  $a_d \neq 0$ . يكون كثير حدود موجبًا بالمقارنة إذا وفقط إذا كان  $a_d > 0$ . وإذا كان  $p(n)$  كثير حدود موجبًا بالمقارنة من الدرجة  $d$ ، فإن  $p(n) = \Theta(n^d)$ . وإذا كان الثابت الحقيقي  $a \geq 0$ ، فإن الدالة  $n^a$  تكون متزايدة باطراد، وإذا كان الثابت الحقيقي  $a \leq 0$ ، فإن الدالة  $n^a$  تكون متناقصة باطراد. ونقول عن دالة ما  $f(n)$  إنها **محدودة بكثير حدود** *polynomially bounded* إذا كان  $f(n) = O(n^k)$  حيث  $k$  ثابت ما.

### الأسيات

إذا كانت  $a > 0$  و  $m$  و  $n$  أعدادًا حقيقية، فتكون لدينا علاقات المساواة التالية:

$$\begin{aligned} a^0 &= 1 , \\ a^1 &= a , \\ a^{-1} &= 1/a , \\ (a^m)^n &= a^{mn} , \\ (a^m)^n &= (a^n)^m , \\ a^m a^n &= a^{m+n} . \end{aligned}$$

وأيًا كان  $n$  و  $a \geq 1$ ، فإن الدالة  $a^n$  متزايدة باطراد في  $n$ . وحيث يكون ذلك مناسبًا، سنفترض أن  $0^0 = 1$ .

يمكن الربط بين معدلات نمو كميات الحدود والدوال الأسية بالحقيقة التالية: أيًا كان الثابتان الحقيقيان  $a$  و  $b$  حيث  $a > 1$ ، فإن:

$$\lim_{n \rightarrow \infty} \frac{n^b}{a^n} = 0. \quad (10.3)$$

وهذا ما نسمع لنا باستنتاج أن

$$n^b = o(a^n).$$

وهكذا، فإن أية دالة أسية ذات أساس أكبر تمامًا من 1 تتزايد بسرعة أكبر من أي كثير حدود.

باستخدام الرمز  $e$  للإشارة إلى العدد  $2.71828\dots$ ، الذي هو أساس دالة اللغاريتم الطبيعي، وأيًا كان العدد الحقيقي  $x$ ، يكون لدينا

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots = \sum_{i=0}^{\infty} \frac{x^i}{i!}, \quad (11.3)$$

حيث ترمز "!" إلى دالة العامل التي سنعرّفها لاحقًا في هذا للقطع. وأيًا كان العدد الحقيقي  $x$ ، تكون لدينا المتراجحة

$$e^x \geq 1 + x, \quad (12.3)$$

وتتحقق المساواة فقط عندما  $x = 0$ . أما إذا كان  $|x| \leq 1$ ، فلدينا التقريب

$$1 + x \leq e^x \leq 1 + x + x^2. \quad (13.3)$$

وعندما  $x \rightarrow 0$ ، فإن تقريب  $e^x \approx 1 + x$  جيد كفاية:

$$e^x = 1 + x + \Theta(x^2).$$

(استخدمنا التدوين المقارب في هذه للمعادلة لوصف السلوك الحدي عندما  $x \rightarrow 0$  بدلاً من  $x \rightarrow \infty$ ). ويكون لدينا:

$$\lim_{n \rightarrow \infty} \left(1 + \frac{x}{n}\right)^n = e^x. \quad (14.3)$$

لجميع قيم  $x$ .

## اللغاريتمات

سنستخدم التدوينات التالية:

$$\lg n = \log_2 n, \quad (\text{لغاريتم اثنائي})$$

$$\ln n = \log_e n, \quad (\text{لغاريتم طبيعي})$$

$$\lg^k n = (\lg n)^k, \quad (\text{رفع إلى أس})$$

$$\lg \lg n = \lg (\lg n). \quad (\text{تركيب})$$

سنستخدم فيما بعد اصطلاحاً تدوينياً هاماً، وهو أن الدوال اللغاريتمية تُطبق فقط على الحد التالي لها مباشرة في أية صيغة، أي إن  $\lg n + k$  تعني  $(\lg n) + k$  وليس  $\lg(n + k)$ . إذا كان  $b > 1$  ثابتاً، فإن الدالة  $\log_b n$  تكون متزايدة تماماً لجميع قيم  $n > 0$ .

إذا كانت  $a > 0$  و  $a \neq 1$  و  $c > 0$  و  $c \neq 1$  أعداداً حقيقية، فإن:

$$\begin{aligned} \log_b a &= \frac{\log_c a}{\log_c b}, \\ \log_c (ab) &= \log_c a + \log_c b, \\ \log_b a^n &= n \log_b a, \\ \log_b (1/a) &= -\log_b a, \\ \log_b a &= \frac{1}{\log_a b}, \\ a^{\log_b c} &= c^{\log_b a}. \end{aligned} \quad (15.3)$$

وذلك عندما يكون أساس اللغاريتم في أيٍّ من المعادلات السابقة لا يساوي 1.

اعتماداً على المعادلة (15.3)، نجد أن تغيير أساس اللغاريتم من ثابت إلى آخر يغيّر قيمة اللغاريتم بعامل ثابت فقط، ولهذا فإننا نستخدم غالباً الرمز " $\lg n$ " عندما لا نهتم بالعوامل الثابتة، كما هي الحال في التدوين- $O$ . ويرى المعلوماتيون أن 2 هو الأساسي الطبيعي الأنسب للغارتمات، لأن العديد من الخوارزميات وبنى المعطيات تتضمن تفريق المسائل إلى جزأين.

هناك نشر بسيط للسلسلة  $\ln(1+x)$  عندما  $|x| < 1$  هو:

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} - \dots$$

وإذا كانت  $x > -1$ ، فلدينا أيضاً المتراجحتان التاليتان:

$$\frac{x}{1+x} \leq \ln(1+x) \leq x, \quad (17.3)$$

وهذه المساواة تتحقق إذا كانت  $x = 0$  فقط.

نقول عن دالة  $f(n)$  إنها محدودة بكثير لغاريتمياً **Polylogarithmically bounded** إذا كان  $f(n) = O(\lg^k n)$  حيث  $k$  ثابت ما. يمكن الربط بين نمو كثيرات الحدود وكثيرات اللغاريتمات بوضع  $\lg n$  بدلاً من  $n$ ، و  $2^n$  بدلاً من  $n$  في المعادلة (10.3) فنجد:

$$\lim_{n \rightarrow \infty} \frac{\lg^b n}{(2^n)^{\lg n}} = \lim_{n \rightarrow \infty} \frac{\lg^b n}{n^a} = 0 .$$

ويمكن أن نستنتج من هذه النهاية أن:

$$\lg^b n = o(n^a)$$

أيًا كان الثابت  $a > 0$ ؛ أي إن أية دالة كثير حدود موجبة تنمو أسرع من أية دالة كثير لغاريتمي.

### العمليات

يُعرّف الرمز  $n!$  [ويقرأ "n عاملي"] للأعداد الطبيعية  $n \geq 1$  كالآتي:

$$n! = \begin{cases} 1 & \text{if } n = 0 , \\ n \cdot (n-1)! & \text{if } n > 0 . \end{cases}$$

أي إن:  $n! = 1 \cdot 2 \cdot 3 \cdots n$ .

هناك حدّ أعلى ضعيف لدالة العامل هو:  $n! \leq n^n$ ، إذ إن كل حدّ في جداء العامل هو على الأكثر  $n$ .

ويعطى تقريب ستيرلنج *Stirling's approximation* بالصيغة الآتية:

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + O\left(\frac{1}{n}\right)\right) , \quad (18.3)$$

حيث  $\blacksquare$  هي أساس المقارنات الطبيعية. وهذا التقريب يعطينا حدًا أعلى أكثر التصاقًا بالعامل، إضافة إلى حدّ أدنى. ويمكننا برهان ما يلي: (انظر التمرين 2.3-3)

$$n! = o(n^n) ,$$

$$n! = \omega(2^n) ,$$

$$\lg(n!) = \Theta(n \lg n) . \quad (19.3)$$

حيث يسمح تقريب ستيرلنج برهان العلاقة (19.3). هذا وتحقق المعادلة التالية أيضًا أيًا كانت  $n \geq 1$ :

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n e^{\alpha_n} \quad (20.3)$$

حيث

$$\frac{1}{12n+1} < \alpha_n < \frac{1}{12n} . \quad (21.3)$$

### التكرار الدالي

نستخدم التدوين  $f^{(l)}(n)$  للتعبير عن تطبيق الدالة  $f(n)$  تكرارًا  $l$  مرة على قيمة بدئية  $l$ . فإذا كانت  $f(n)$  دالة معرفة على الأعداد الحقيقية، وكان  $l$  عددًا طبيعيًا، فإننا نعرّف هذا التدوين غرديًا على النحو الآتي:



$$f^{(i)}(n) = \begin{cases} n & \text{if } i = 0, \\ f(f^{(i-1)}(n)) & \text{if } i > 0. \end{cases}$$

فمثلاً، إذا كان  $f(n) = 2n$ ، فإن  $f^{(i)}(n) = 2^i n$ .

### دالة اللغاريتم المكرر

نستخدم التدوين  $\lg^* n$  [ويقرأ "لوغ بحمة لـ  $n$ "] لتشير إلى دالة اللغاريتم المكرر المعروفة كالتالي: لتكن  $\lg^{(i)} n$  معرفة وفق التعريف السابق، مع الدالة  $\lg n = f(n)$ . ولما كان لغاريتم الأعداد السالبة غير معرف، فإن الدالة تكون معرفة إذا كان  $\lg^{(i-1)} n > 0$  فقط. تأكد أنك تُعزِّز بين  $\lg^{(i)} n$  (وهي دالة اللغاريتم مطبقة  $i$  مرة على التوالي، بدءاً من المحدد  $n$ )، وبين  $\lg^i n$  (وهو لغاريتم  $n$  مرفوعاً إلى القوة  $i$ ). وبذلك نعرِّف دالة اللغاريتم المكرر كالتالي:

$$\lg^* n = \min\{i \geq 1 : \lg^{(i)} n \leq 1\}$$

إن اللغاريتم المكرر دالة متزايد ببطء شديد:

$$\lg^* 2 = 1,$$

$$\lg^* 4 = 2,$$

$$\lg^* 16 = 3,$$

$$\lg^* 65536 = 4,$$

$$\lg^*(2^{65536}) = 5.$$

ولما كان عدد الذرات في العالم الممكن ملاحظته يُقدَّر بنحو  $10^{80}$ ، وهو أقل بكثير من  $2^{65536}$ ، فمن النادر أن نقع على مُدخل حجمه  $n$  بحيث يكون  $\lg^* n > 5$ .

### أعداد فيبوناتشي

نعرِّف أعداد فيبوناتشي *Fibonacci numbers* بالعلاقة التّعددية التالية:

$$F_0 = 0$$

$$F_1 = 1$$

(22.3)

$$F_i = F_{i-1} + F_{i-2} \quad \text{for } i \geq 2.$$

وهكذا، فإن أي عدد من أعداد فيبوناتشي هو مجموع العددين اللذين يسبقانه، وهذا ما يعطي المتتالية

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \dots$$

إن أعداد فيبوناتشي مرتبطة بالكسر الذهبي *golden ratio*،  $\phi$ ، ويراافقه  $\hat{\phi}$ ، وهما جذرا المعادلة

$$x^2 = x + 1$$

(23.3)

وهما معرفان بالصيغتين التاليتين (انظر التمرين 2.3-6):

$$\phi = \frac{1 + \sqrt{5}}{2}$$

$$= 1.61803 \dots$$

$$\hat{\phi} = \frac{1 - \sqrt{5}}{2}$$

$$= -0.61803 \dots$$

وتحديداً لدينا

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}},$$

وهذا ما يمكن برهانه بالاستقراء (التمرين 2.3-7). ولما كان  $|\hat{\phi}| < 1$ ، فلدينا

$$\frac{|\hat{\phi}^i|}{\sqrt{5}} < \frac{1}{\sqrt{5}}$$

$$< \frac{1}{2},$$

وهذا يقتضي أن

$$F_i = \left\lfloor \frac{\phi^i}{\sqrt{5}} + \frac{1}{2} \right\rfloor, \quad (25.3)$$

أي إن عدد فيبوناتشي  $F_i$  ذي الرقم  $i$ ، يساوي  $\phi^i/\sqrt{5}$  مدوّراً إلى أقرب عدد طبيعي. إذن، فإن أعداد فيبوناتشي تتزايد أسياً.

تعاريف

### 1-2.3

برهن أنه إذا كانت  $f(n)$  و  $g(n)$  دالتين متزايدتين باطراد، فإن  $f(n) + g(n)$  و  $f(g(n))$  متزايدتان باطراد. وإذا كانت  $f(n)$  و  $g(n)$  دالتين متزايدتين باطراد وموجبتين، فإن  $f(n) \cdot g(n)$  متزايدة باطراد أيضاً.

### 2-2.3

برهن للمعادلة (16.3).

### 3-2.3

برهن للمعادلة (19.3). وبرهن أيضاً أن  $n! = \Theta(2^n)$  وأن  $n! = o(n^n)$ .

### ■ 4-2.3

هل الدالة  $[lg n]!$  محدودة بكثير حدود؟ وهل الدالة  $[lg lg n]!$  محدودة بكثير حدود؟

### ■ 5-2.3

أي دالة أكبر بالمقارنة:  $lg(lg^* n)$  أم  $lg^*(lg n)$ ؟

### 6-2.3

يَبَيِّنْ أَنَّ الْكُسْرَ الذَّهَبِيَّ  $\phi$  وَمُرَافِقَهُ  $\hat{\phi}$  يَحَقِّقَانِ مَعًا لِمُعَادَلَةِ  $x^2 = x + 1$ .

### 7-2.3

بَرِّهْنِ بِالِاسْتِقْرَاءِ أَنَّ عَدَدَ فَيُونَاتَشِي ذَا الرِّفْعِ  $i$  يَحَقِّقُ الْمُسَاوَاةَ:

$$F_i = \frac{\phi^i - \hat{\phi}^i}{\sqrt{5}},$$

حَيْثُ  $\phi$  هُوَ الْكُسْرُ الذَّهَبِيَّ وَ  $\hat{\phi}$  مُرَافِقُهُ.

### 8-2.3

بَرِّئْ أَنَّ  $k \ln k = \Theta(n)$  يَتَّخِذُ  $k = \Theta(n / \ln n)$ .

## مَسَائِلُ

### 1-3 السلوك المقارب لكثيرات الحدود

ليكن

$$p(n) = \sum_{i=0}^d a_i n^i$$

حَيْثُ  $a_d > 0$ ، كَثِيرٌ حَدُودٌ فِي  $n$  مِنَ الدَّرَجَةِ  $d$ ، وَلِيَكُنْ  $k$  ثَابِتًا مَا. اسْتَخْدِمِ تَعَارِيفَ التَّدْوِينَاتِ الْمُقَارِبَةِ لِلدَّرْجَانِ عَلَى الْخُصَالِصِ التَّالِيَةِ:

أ. إِذَا كَانَ  $k \geq d$ ، فَإِنَّ  $p(n) = O(n^k)$ .

ب. إِذَا كَانَ  $k \leq d$ ، فَإِنَّ  $p(n) = \Omega(n^k)$ .

ت. إِذَا كَانَ  $k = d$ ، فَإِنَّ  $p(n) = \Theta(n^k)$ .

ث. إِذَا كَانَ  $k > d$ ، فَإِنَّ  $p(n) = o(n^k)$ .

ج. إِذَا كَانَ  $k < d$ ، فَإِنَّ  $p(n) = \omega(n^k)$ .

### 2-3 النمو المقارب التسي

فِي الْجَدُولِ الْآتِي لَدَيْنَا أَزْوَاجُ الْعِبَارَاتِ  $(A, B)$ ، يَبَيِّنْ هَلْ هِيَ  $O$  أَوْ  $\omega$  أَوْ  $\Omega$  أَوْ  $\Theta$  أَوْ  $B$ ? افترضْ أَنَّ  $k \geq 1$  وَ  $\epsilon > 0$  وَ  $c > 1$  ثَابِتًا. يَجِبُ أَنْ يَكُونَ جَوَابُكَ بِصِيغَةِ "نَعَمْ" أَوْ "لَا" فِي كُلِّ خَانَةٍ مِنْ خَانَاتِ الْجَدُولِ.

$\Theta$	$\omega$	$\Omega$	$o$	$O$	$B$	$A$
					$n^\epsilon$	$\lg^k n$ أ.
					$c^n$	$n^k$ ب.
					$n^{\sin n}$	$\sqrt{n}$ ت.
					$2^{n/2}$	$2^n$ ث.
					$c^{\lg n}$	$n^{\lg c}$ ج.
					$\lg(n^n)$	$\lg(n!)$ ح.

### 3-3 الترتيب وفق معدلات النمو المقاربة

أ. رتب الدوال التالية وفق ترتيب نموها؛ أي أوجد الترتيب  $g_1, g_2, \dots, g_{30}$  الذي يحقق:  
 $g_1 = \Omega(g_2)$ ,  $g_2 = \Omega(g_3)$ , ...,  $g_{29} = \Omega(g_{30})$ .  
 فتمت فالتك إلى صفوف تكافؤ بحيث تكون الدالتان  $f(n)$  و  $g(n)$  في الصف نفسه إذا فقط إذا كان  $f(n) = \Theta(g(n))$ .

$\lg(\lg^* n)$	$2^{\lg^* n}$	$(\sqrt{2})^{\lg n}$	$n^2$	$n!$	$(\lg n)!$
$\left(\frac{3}{2}\right)^n$	$n^3$	$\lg^2 n$	$(\lg n!)$	$2^{2^n}$	$n^{1/\lg n}$
$\ln \ln n$	$\lg^* n$	$n \cdot 2^n$	$2^{\lg \lg n}$	$\ln n$	1
$2^{\lg n}$	$(\lg n)^{\lg n}$	$e^n$	$4^{\lg n}$	$(n+1)!$	$\sqrt{\lg n}$
$\lg^*(\lg n)$	$2^{\sqrt{2} \lg n}$	■	$2^n$	$n \lg n$	$2^{2^{n+1}}$

ب. أعط مثالاً لدالة موجبة واحدة  $f(n)$  بحيث لا تحقق  $O(g_1(n))$  ولا  $\Omega(g_1(n))$  مع أي من الدوال  $g_1(n)$  الواردة في الجزء السابق (أ).

### 4-3 خواص التدوين المقارب

لتكن  $f(n)$  و  $g(n)$  دالتين موجبتين بالمقاربة. برهن صحة أو عدم صحة كل من للمخمنات التالية:

أ.  $f(n) = O(g(n))$  يقتضي  $g(n) = O(f(n))$ .

ب.  $f(n) + g(n) = \Theta(\min(f(n), g(n)))$ .

ت.  $f(n) = O(g(n))$  يقتضي  $\lg(f(n)) = O(\lg(g(n)))$ ، حيث  $\lg(g(n)) \geq 1$  و  $f(n) \geq 1$  لكل قيم  $n$  الكبيرة كفاية.

ث.  $f(n) = O(\lg(n))$  يقتضي  $2^{f(n)} = O(2^{g(n)})$ .

ج.  $f(n) = O((f(n))^2)$ .

ح.  $f(n) = O(g(n))$  يقتضي  $g(n) = \Omega(f(n))$ .

خ.  $f(n) = \Theta(f(n/2))$ .

د.  $f(n) + o(f(n)) = \Theta(f(n))$ .

### 5-3 أشكال معدلة من $O$ و $\Omega$

يُعرف بعض المؤلفين  $\Omega$  بطريقة مختلفة قليلاً عن الطريقة التي عرّفناها بها هنا. نستخدم لهذا التعريف البديل الرمز  $\bar{\Omega}$  (ويقرأ "أوميجا لأعماة")، ونقول إن  $f(n) = \bar{\Omega}(g(n))$  إذا وُجد ثابت موجب  $c$  بحيث يكون  $f(n) \geq cg(n)$  لعددٍ لا نهائيٍّ من القيم الصحيحة  $n$ .

أ. يَبْينُ أنه إذا كانت الدالتان  $f(n)$  و  $g(n)$  للموجبتان بالمقارنة، فإما أن يكون  $f(n) = O(g(n))$ ، وإما أن يكون  $f(n) = \bar{\Omega}(g(n))$ ، وإما يكونا متساويين على حين أن هذا لا يكون صحيحاً إذا وضعنا  $\Omega$  مكان  $\bar{\Omega}$ .

ب. اشرح المحاسن والمساوئ الكامنة في استخدام  $\bar{\Omega}$  مكان  $\Omega$  في توصيف أزمنة تنفيذ البرامج.

يُعرف بعض المؤلفين  $O$  أيضاً بطريقة مختلفة قليلاً. نستخدم لهذا التعريف البديل الرمز  $O'$ ، ونقول إن  $f(n) = O'(g(n))$  إذا وفقط إذا  $|f(n)| = O(g(n))$ .

ث. ما الذي يطرأ على اتجايفي الاقتضاء "إذا وفقط إذا" في المبرهنة 1.3 إذا وضعنا  $O'$  مكان  $O$ ، وحافظنا على  $\Omega$ ؟

يستعمل بعض المؤلفين الرمز  $\tilde{O}$  (ويقرأ "Soft-oh")، للدلالة على  $O$  مع إهمال العوامل اللغاريتمية، ويعرفونه كما يلي:

$\tilde{O}(g(n)) = \{f(n) : \text{there exist positive constants } c, k, \text{ and } n_0 \text{ such that}$

$$\| \leq f(n) \leq cg(n) \lg^k(n) \text{ for all } n \geq n_0 \}.$$

ث. عرّف بأسلوب مماثل كلاً من:  $\tilde{\Omega}$  و  $\tilde{\Theta}$ ، ثم برهن المبرهنة 1.3 باستخدام هذه التديونات المعدلة.

### 6-3 الدوال المكررة

يمكن تطبيق عملية التكرار " المستخدمة في دالة  $\lg$  على أية دالة متزايدة بانتظام  $f(n)$  معرفة على الأعداد الحقيقية. فإذا كان الثابت  $c \in \mathbb{R}$ ، فإننا نعرف الدالة المكررة  $f_c^*$  على أنها

$$f_c^*(n) = \min\{i \geq 0 : f^{(i)}(n) \leq c\},$$

والتي ليس من الضروري أن تكون معرفة في كل الحالات. وبعبارة أخرى: إن قيمة  $f_c^*(n)$  هي عدد مرات التطبيق المتتالي للدالة  $f$  اللازم لتتناقص قيمتها حتى  $c$  أو أقل.

أعط لكل من الدوال  $f(n)$  التالية والثوابت  $c$ ، حدًا محكمًا قدر الإسكان على  $f_c^*(n)$ .

$f_c^*(n)$	$c$	$f(n)$
أ.	$\infty$	$n - 1$
ب.	1	$\lg n$
ت.	1	$n/2$
ث.	2	$n/2$
ج.	2	$\sqrt{n}$
ح.	1	$\sqrt{n}$
خ.	2	$n^{1/3}$
د.	2	$n/\lg n$

### ملاحظات الفصل

يميد Knuth [209] أصل تدوين- $O$  إلى نصر في نظرية الأعداد بقلم P.Bachmann يعود إلى العام 1892. وقد ابتدع E.Landau تدوين- $\Theta$  في عام 1909 ليستخدمه في مناقشته توزيع الأعداد الأولية. وقد ابتدع Knuth [213] تدويني  $\Omega$  و  $\Theta$  ليصحح الاستخدام الشائع في الأدبيات، رغم أنه غير دقيق تقنيًا، لتدوين- $O$  لحدود عليا ودنيا على السواء. وما زال الكثير يستخدمون تدوين- $O$  حيث إن تدوين- $\Theta$  هو أكثر دقة تقنيًا. المزيد من المناقشة حول تاريخ وتطور التدوينات للمقارنة موجود في Knuth [209, 213] وفي Brassard و Bratley [55].

لا يُجمع المؤلفون على تعريف التدوينات للمقاربة بالطريقة نفسها، إلا أن التعاريف المختلفة تتفق في معظم الحالات الشائعة. تشمل بعض التعاريف دوال غير موجبة بالمقاربة، مادامت قيمها المطلقة محدودة كما ينبغي. تُنسب للمعادلة (20.3) إلى Robbins [297]. ويمكن العثور على خواص أخرى للدوال الرياضية الأساسية في أي مرجع جيد في الرياضيات، مثل Abramowitz و Stegun [1] أو Zwillinger [362]، أو في كتاب في حساب التفاضل والتكامل مثل Apostol [18] أو Thomas وآخرون [334]. ويضم كلٌّ من Knuth [209] و Graham و Knuth و Patashnik [152] مادةً غنيةً في الرياضيات المتقطعة المستخدمة في علوم الحاسوب.

رأينا في المقطع 1.3.2 أن الفرز بالدمج يقدم مثالا عن نموذج خوارزميات فرق-تسد. تذكر أننا في سياق فرق-تسد، نحل مسألة ما عُدديًا، بتطبيق ثلاث خطوات في كل مستوى من العُدديّة:

فرق المسألة إلى عدد من المسائل الجزئية التي هي متسخت أصغر من المسألة نفسها.

نُد أي سطر على المسائل الجزئية بحلّها عُدديًا. في حال كانت حجوم المسائل الجزئية صغيرة كفاية، يمكنك أن تحلّها بطريقة مباشرة.

جُمع حلول المسائل الجزئية في حل للمسألة الأصلية.

عندما تكون حجوم المسائل الجزئية كبيرة كفاية لحلّها عُدديًا، فإننا ندعو ذلك *بالحالة القُودية recursive case*. ما إن تصبح المسائل الجزئية صغيرة كفاية بحيث نتوقف عن تقسيمها عُدديًا، نقول إن القُودية قد وصلت إلى "القاع"، وإننا قد نزلنا إلى *الحالة الأساسية base case*. قد نضطر، في بعض الأحيان - إضافة إلى حل المسائل الجزئية التي هي متسخت أصغر من المسألة نفسها - إلى حل مسائل جزئية لا تشبه تمامًا المسألة الأصلية. نفدّ حل مثل هذه المسائل الجزئية جزءًا من خطوة التجميع.

سنطلع في هذا الفصل على المزيد من الخوارزميات المعتمدة على طريقة فرق-تسد. الخوارزمية الأولى نحل مسألة الصفيقة الجزئية العظمى: وفيها يكون دخل الخوارزمية صفيقة من الأعداد، وتحدّد الصفيقة الجزئية المتتالية ذات المجموع الأكبر. ثم نطلع على خوارزمية فرق-تسد لحساب جداء مصفوفات  $n \times n$ . تُنفذ الأولى في زمن  $\Theta(n^3)$ ، أي إننا لا تتفوق على الطريقة المباشرة في جداء المصفوفات الطريقة، فيما تُنفذ الثانية، خوارزمية شتراسن Strassen، في زمن  $\Theta(n^{2.81})$ ، وهذا الزمن يتفوق على زمن الطريقة المباشرة بالمقارنة.

### العلاقات القُودية

تقرن العلاقات القُودية بطريقة فرق-تسد لأنها تعطينا طريقة طبيعيّة لتوصيف أزمنة تنفيذ الخوارزميات التي تتبع هذه الطريقة. *العلاقة القُودية recurrence* هي معادلة أو متراجحة توصف دالة بدلالة قيمها نفسها لمُدخلات أصغر منها. رأينا مثلاً، في المقطع 2.3.2، أنه يمكن توصيف  $T(n)$  زمن تنفيذ إجرائية



MERGE-SORT في أسوأ الحالات باستخدام العلاقة العودية

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1, \end{cases} \quad (1.4)$$

وذكرنا أن حلها هو  $T(n) = \Theta(n \lg n)$ .

يمكن أن نأخذ العلاقات العودية صيغاً عديدة. فعلى سبيل المثال قد تقسم خوارزمية عودية المسألة إلى مسائل جزئية من أحجام غير متساوية، كأن نفرقها مثلاً  $2/3$  إلى  $1/3$ . إذا كانت خطوات التقسيم والتجميع تستغرق زمناً خطياً، فيستنتج عن هذه الخوارزمية العلاقة العودية  $T(n) = (2n/3) + T(n/3) + \Theta(n)$ . ليس من الضروري أن تكون المسائل الجزئية ذات أجزاء ثابتة من حجم المسألة الأصلية. فمثلاً، قد نقوم نسخة عودية من البحث الخطي (انظر التمرين 1.2-3) بتوليد مسألة جزئية تحتوي على عدد من العناصر يقل عن المسألة الأصلية بواحد فقط. ويستغرق كل استدعاء عودي زمناً ثابتاً إضافة إلى زمن الاستدعاءات العودية التي يقوم بها، وهذا يعطي العلاقة العودية  $T(n) = T(n-1) + \Theta(1)$ .  
يقدم هذا الفصل ثلاث طرق لحل العلاقات العودية- أي للحصول على حدود مقارنة لحلها معبر عنها بـ "O" أو "Θ".

- **في طريقة التعويض substitution method**، نأخذ حدًا ونستخدم الاستقراء الرياضي لنبرهن على صحة تخميننا.
- نحول طريقة شجرة العودية recursion-tree العلاقة العودية إلى شجرة، تمثل عقدها التكاليف الواجبة في مختلف مستويات العودية، ثم نستخدم تقنيات لحّد سلاسل الجمع في حل العلاقة العودية.
- نقدم الطريقة الرئيسة master method حدودًا للعلاقات العودية من الصيغة

$$T(n) = aT(n/b) + f(n), \quad (2.4)$$

حيث  $a \geq 1$ ، و  $b > 1$ ، و  $f(n)$  دالة مطّاة. تصادفنا مثل هذه العلاقات العودية مراراً. فعلاقة عودية مثل تلك في المعادلة (2.4) توصف خوارزمية فرز-تسد التي تنشئ  $a$  مسألة جزئية، حجم كل منها  $1/b$  من حجم المسألة الأصلية، وتستغرق فيها خطوات التقسيم والتجميع معاً زمناً  $f(n)$ . ويطلب استخدام الطريقة الرئيسة أن نحفظ ثلاث حالات في ذاكرتك، ولكن ما إن تتمكن من حفظها، حتى يصبح تحديد حدود مقارنة العديد من العلاقات العودية البسيطة سهلاً عليك. سنستخدم الطريقة الرئيسة في تحديد أزمنة تنفيذ خوارزميات فرز-تسد لمسألة الصفيقة الجزئية العظمى ولجداء المصفوفات، ولخوارزميات أخرى موجودة في هذا الكتاب تعتمد على مبدأ فرز-تسد.

ستصادفنا أحياناً علاقات عودية بصيغة متراجحات لا بصيغة علاقات مساواة، مثل  $T(n) \leq 2T(n/2) + \Theta(n)$ . ولما كانت مثل هذه العلاقات تعطي حدًا أعلى لـ  $T(n)$  فقط، فإننا

منصوغ حلها باستخدام تدوين-0 بدلاً من تدوين- $\Theta$ . وبالمثل، إذا عكست المتراجحة لتصبح  $T(n) \geq 2T(n/2) + \Theta(n)$ ، فنستخدم تدوين- $\Omega$  في حلها لأنها أصلاً تعطي حدًا أدنى لـ  $T(n)$  فقط.

### تفاصيل تقنية في العلاقات القُودية

عمليًا، عندما نستعرض علاقات غُودية ونحلها، فإننا نعمل بعض التفاصيل التقنية. على سبيل المثال، إذا استدعينا MERGE-SORT لفرز  $n$  عنصرًا، عندما يكون  $n$  فرديًا، فإننا نحصل على مسألتين حجمهما  $\lfloor n/2 \rfloor$  و  $\lceil n/2 \rceil$ . وأيًا من هاتين القيمتين لا تساوي فعلاً  $n/2$ ، لأن  $n/2$  ليس عددًا طبيعيًا عندما يكون  $n$  فرديًا. إن العلاقة التي تصف رياضياً زمن تنفيذ إجرائية MERGE-SORT في أسوأ الحالات هو فعليًا:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n) & \text{if } n > 1 \end{cases} \quad (3.4)$$

ومثل الشروط المحددة خطأً آخر من التفاصيل التي نتجاهلها عادة. ولما كان زمن تنفيذ خوارزمية ما -دخلي ذي حجم ثابت- ثابتًا أيضًا، فإن صيغة العلاقات القُودية الناتجة عن أزمة تنفيذ الخوارزميات هي  $T(n) = \Theta(1)$  عمومًا، حيث  $n$  صغيرة كفاية. ولهذا، ولغرض التبسيط، سنعمل عمومًا عبارات الشروط المحددة للعلاقات القُودية، وسنفترض أن  $T(n)$  ثابتة عندما تكون  $n$  صغيرة. فعلى سبيل المثال، نكتب عادة العلاقة القُودية في (1.4) بالصيغة

$$T(n) = 2T(n/2) + \Theta(n) \quad (4.4)$$

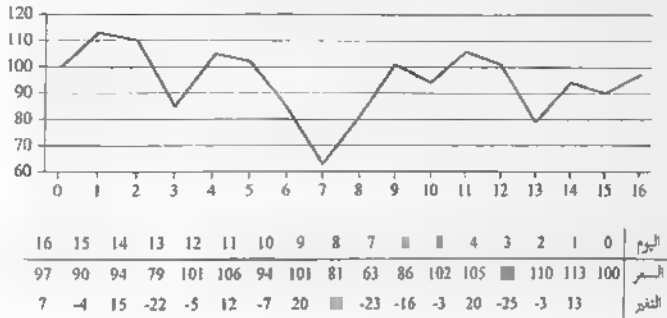
دون أن نوضح صراحة القيم عندما تكون  $n$  صغيرة. والسبب هو أنه على الرغم من تغير حل العلاقة القُودية بتغير قيمة  $T(1)$ ، فإن الحل لا يتغير عادة إلا بعامل ثابت، وهكذا تبقى مرتبة النمو نفسها دون تغير.

عندما نعطي العلاقات القُودية ونحلها، فغالبًا ما نحذف الأرضيات والأسقف والشروط المحددة. إننا نتابع قُدُما دون هذه التفاصيل، ثم نقرر لاحقًا: هل هي مهمة أم لا؟ ومع أنها غالبًا ما تكون غير مهمة، فينبغي معرفة متى تكون مهمة فعلاً. وتساعد على ذلك الخبرة، إضافة إلى بعض المبرهنات التي تشير إلى أن هذه التفاصيل لا تؤثر في الحدود المقاربة لكثير من العلاقات القُودية التي تصادفنا عند تحليل خوارزميات فرق-تسد (انظر المبرهنة 1.4). على أية حال، سنعالج في هذا الفصل بعض هذه التفاصيل ونبين النقاط الدقيقة المتعلقة بطرق حل العلاقات القُودية.

## مسألة الصيغة الجزئية العظمى

1.4

افترض أنه أتاحت لك الفرصة أن تستثمر في شركة Volatile Chemical Corporation. وكما هي حال المواد الكيميائية الطيارة التي تنتجها الشركة، فإن سعر سهم الشركة طيار أيضًا ومتذبذب. يُسمح لك أن تشتري وحدة من الأسهم مرة واحدة فقط لتبيعها في موعد لاحق، على أن يتم البيع والشراء بعد إغلاق



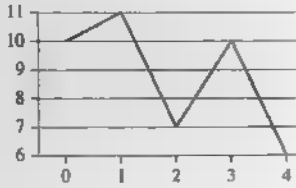
**الشكل 1.4** معلومات عن سعر سهم شركة Volatile Chemical Corporation بعد إغلاق المبادلات خلال مدة 17 يوماً. يبيّن محور الأفقي للمخطط البياني اليوم، ويبيّن محور العمودي السعر. وبين السطر الأسفل في الجدول تحت للمخطط تغير السعر مقارنة باليوم السابق.

المداولات في غاية اليوم. للتعويض عن هذا القيد، يُسمح لك أن تعرف سعر السهم في المستقبل. هدفك هو الحصول على الربح الأعظم. يبيّن الشكل 1.4 سعر السهم خلال مدة طولها 17 يوماً. يمكنك الشراء، في أي وقت ترده ولمرة واحدة، بعد اليوم 0 الذي يكون فيه سعر السهم \$100. ستعجب طبقاً في أن "تشتري بسعر منخفض، وتبيع بسعر مرتفع" - بالأحرى أن تشتري بأخفض سعر ممكن، ثم تبيع بعد ذلك بأعلى سعر ممكن - وذلك لتعظم ربحك. ولكن لسوء الحظ، قد لا يكون بإمكانك فعلاً الشراء بأدنى سعر ممكن ثم البيع بأعلى سعر ممكن خلال مدة معطاة. ففي الشكل 1.4 نرى أن أدنى سعر يتحقق بعد اليوم السابع، وهو يلي اليوم الأول الذي يصل فيه السعر إلى أعلى قيمة.

قد تفكر في أنه بإمكانك أن تعظم الربح سواء بالشراء دائماً بأخفض سعر أو بالبيع دائماً بأعلى سعر. على سبيل للمثال، قد نعظم الربح بالشراء بالسعر الأدنى، بعد اليوم 7. لو نجحت هذه الاستراتيجية دائماً، لكان من السهل تحديد طريقة لتعظيم الربح: أوجد أعلى سعر وأدنى سعر، ثم عُدْ نحو اليسار بدءاً من أعلى سعر لتحدد أدنى سعر سابق له، أو ابدأ من أدنى سعر وتقدمْ ميّناً لتجد أعلى سعر لاحق له، ثم اختر زوج الأسعار الذي يحقق أكبر فارق. يبيّن الشكل 2.4 مثلاً معاكساً بسيطاً يبيّن أن الربح الأعظم أحياناً لا يأتي بالشراء عند أدنى سعر ولا بالبيع عند أعلى سعر.

### حل فج

يمكننا بسهولة إعطاء حلّ فجّ لهذه المسألة: يكفي أن تجرّب كل زوجين من أيام شراء ومبيع يكون فيه يوم الشراء سابقاً ليوم البيع. يمكن في مدة من  $n$  يوماً إيجاد  $\binom{n}{2}$  زوجاً من هذه الأيام. ولما كان  $\binom{n}{2}$  من رتبة



اليوم	0	1	2	3	4
السعر	10	11	7	10	6
التغير		1	-4	3	-4

**الشكل 2.4** مثال يبين أن الربح الأعظم لا يتحقق دائماً بالبدء بالسعر الأدنى أو بالانتهاء بالسعر الأعلى. مرة ثانية، يشير المحور الأفقي إلى اليوم، وبين المحور العمودي السعر. الربح الأعظم هو \$3 للسهم الواحد، ويمكن تحقيقه بالشراء بعد اليوم 2 وبالبيع بعد اليوم 3. ليس السعر \$7 بعد اليوم 2 هو السعر الأدنى عمومًا، ولا السعر \$10 بعد اليوم 3 هو السعر الأعلى عمومًا.

$\Theta(n^2)$ ، وكان أفضل ما يمكن أن نأمله هو أن يستغرق حساب كل زوج من الأيام زمانًا ثابتًا، فإن مثل هذه الطريقة تستغرق زمانًا  $\Omega(n^2)$ . هل بإمكاننا تحسين ذلك؟

### تحويل

بهدف تصميم خوارزمية بزمن تنفيذ  $\Theta(n^2)$ ، سننظر إلى الدنجل بطريقة مختلفة قليلًا. نريد أن نجد متتالية من الأيام يكون الفارق الصافي من أول يوم وحتى آخر يوم أعظمًا. بدلاً من النظر إلى السعر اليومي ننظر إلى التغير اليومي في السعر، حيث يكون التغير في اليوم  $i$  هو الفارق بين سعر نهاية اليوم  $i-1$  وسعر نهاية اليوم  $i$ . يبين الجدول في الشكل 1.4 هذه التغيرات في السطر الأسفل. إذا كنا نتعامل مع هذا السطر على أنه صيغة  $A$  كتلك المبينة في الشكل 3.4، فإن هدفنا الآن هو إيجاد صيغة جزئية متتالية وغير خالية من  $A$  يكون مجموع عناصرها هو الأكبر. نسمي هذه الصيغة الجزئية المتتالية **الصيغة الجزئية العظمى** *maximum subarray*. على سبيل للمثال: في الصيغة المبينة في الشكل 3.4، الصيغة الجزئية العظمى من  $A[1..16]$  هي  $A[8..11]$ ، مع المجموع 43. إذن، سنرغب في الشراء مباشرة قبل اليوم 8 (أي في نهاية اليوم 7) وأن نبيع بعد نهاية اليوم 11، لنحقق بذلك ربحًا يصل إلى \$43 للسهم الواحد.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

صيغة جزئية عظمى

**الشكل 3.4** التغير في أسعار الأسهم باعتباره مسألة صيغة جزئية عظمى. في هذا المثال، تحقق الصيغة الجزئية  $A[8..11]$  ذات المجموع 43، المجموع الأعلى من بين أية صيغة جزئية ملائمة للصيغة  $A$ .

لا يساعدنا هذا التحويل كثيرًا، أول وهلة. فما زلنا بحاجة إلى التحقق من  $\Theta(n^2) = \Theta(n^2)$  صيغة جزئية لـ  $n$  يومًا. يطلب التمرين 4.4-2 إليك أن تبين أنه، على الرغم من أن حساب كلفة صيغة جزئية قد يستغرق زمنًا متناسبًا مع طول هذه الصيغة الجزئية عند حساب كل المجمامع الجزئية وعددها  $\Theta(n^2)$ ، فإن بإمكاننا تنظيم الحسابات بحيث يستغرق حساب كل مجموع صيغة جزئية زمنًا  $O(1)$ ، بمعرفة بمجمامع الصيغيات الجزئية السابقة. وعليه فإن حل الطريقة الفعّلة يستغرق زمنًا  $\Theta(n^2)$ .

فلنبحث إذن عن حل أكثر فعالية لمسألة الصيغة الجزئية العظمى. وعندما نقوم بذلك، فإننا نتحدث عادة عن "صيغة جزئية عظمى" وليس عن "الصيغة الجزئية العظمى"، إذ قد يكون هناك أكثر من صيغة جزئية تحقق المجموع الأعظم.

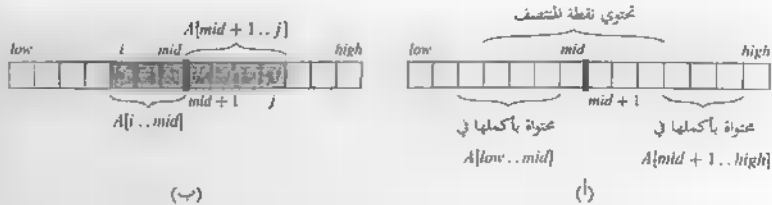
إن مسألة الصيغة الجزئية العظمى تكون مثيرة للاهتمام فقط عندما تحتوي الصيغة بعض الأعداد السالبة. فعندما تكون كل مركبات الصيغة موجبة، فإن حل المسألة لن يكون فيه أي تحدٍّ، إذ إن كامل الصيغة يحقق المجموع الأعلى.

#### حل باستخدام فرق-تسد

دعنا نفكر كيف يمكن أن نحل مسألة الصيغة الجزئية العظمى باستخدام تقنية فرق-تسد. افترض أننا نريد نجد صيغة جزئية عظمى من الصيغة الجزئية  $A[low..high]$ . تقترح طريقة فرق-تسد أن نقسم الصيغة الجزئية إلى صيغتين جزئيتين بطول متساوٍ إذا كان ذلك ممكنًا. أي أن نجد نقطة المنتصف في الصيغة الجزئية، ونسميها  $mid$ ، وأن ندرس الصيغتين الجزئيتين  $A[low..mid]$  و  $A[mid + 1..high]$ . يبين الشكل 4.4 أ) أن أية صيغة جزئية متتالية  $A[i..j]$  من  $A[low..high]$  يجب أن تقع غامًا في أحد المواقع التالية:

- محتواة بكاملها في صيغة الجزئية  $A[low..mid]$ ، بحيث يكون  $low \leq i \leq j \leq mid$ .
- محتواة بكاملها في الصيغة الجزئية  $A[mid + 1..high]$ ، بحيث يكون  $mid < i \leq j \leq high$ ، أو
- تحتوي نقطة المنتصف، بحيث يكون:  $low \leq i \leq mid < j \leq high$ .

إذن، لا يمكن لصيغة جزئية عظمى أن تقع إلا في أحد هذه المواقع. وفي الواقع، يجب أن يكون مجموع صيغة جزئية عظمى من  $A[low..high]$  هو الأكبر بين مجاميع كل الصيغيات الجزئية المحتواة كليًا في  $A[low..mid]$ ، والمحتواة كليًا في  $A[mid + 1..high]$ ، أو التي تحتوي نقطة المنتصف. بإمكاننا أن نجد الصيغتين الجزئيتين العظميين لـ  $A[low..mid]$  و  $A[mid + 1..high]$  عوديًا، لأن هاتين المسألتين الجزئيتين هما متساويان أصغر حجمًا من مسألة إيجاد الصيغة الجزئية العظمى. وبهذا، يكون كل ما يتبقى علينا فعله هو إيجاد الصيغة الجزئية ذات المجموع الأكبر من بين الصيغيات الثلاث التي تحتوي نقطة المنتصف.



الشكل 4.4 (أ) المواقع الممكنة للصفيفات الجزئية من  $A[low..high]$ : محتواة بأكملها في  $A[low..mid]$ ، محتواة بأكملها في  $A[mid + 1..high]$ ، أو تحتوي نقطة المنتصف  $mid$ . (ب) أية صفيفة جزئية من  $A[low..high]$  تحتوي نقطة المنتصف تكون مكونة من صفيقتين جزئيتين  $A[i..mid]$  و  $A[mid + 1..j]$  حيث  $low \leq i \leq mid$  و  $mid < j \leq high$ .

يمكننا بسهولة إيجاد صفيقة جزئية عظمى تحتوي نقطة المنتصف في زمن خطي متناسب مع حجم الصفيقة  $A[low..high]$ . إن هذه المسألة ليست متنسخاً أصغر من مسائلتنا الأصلية، لأن فيها قيداً إضافياً يتمثل في ضرورة أن تضم الصفيقة الجزئية المختارة نقطة المنتصف. وكما يبين الشكل 4.4(ب) فإن أية صفيقة جزئية تحتوي نقطة المنتصف هي في الحقيقة مركبة من صفيقتين جزئيتين  $A[i..mid]$  و  $A[mid + 1..j]$ ، حيث  $low \leq i \leq mid$  و  $mid < j \leq high$ . لذا، فما علينا سوى إيجاد صفيقتين جزئيتين من الشكل  $A[i..mid]$  و  $A[mid + 1..j]$  ثم ضمهما معاً. إن دخل الإجراء FIND-MAX-CROSSING-SUBARRAY هو الصفيقة  $A$  والمؤشرات  $low$  و  $mid$  و  $high$ ، ويعيد ثلاثية تحوي المؤشرين اللذين يحدان صفيقة جزئية عظمى تضم نقطة المنتصف، إضافة إلى مجموع القيم في صفيقة جزئية عظمى.

FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )

```

1  left-sum = -∞
2  sum = 0
3  for i = mid downto low
4      sum = sum + A[i]
5      if sum > left-sum
6          left-sum = sum
7          max-left = i
8  right-sum = -∞
9  sum = 0
10 for j = mid + 1 to high
11     sum = sum + A[j]
12     if sum > right-sum
13         right-sum = sum
14         max-right = j
15 return (max-left, max-right, left-sum + right-sum)

```

يعمل هذا الإجراء كما يلي. نجد السطور 1-7 صفيقة جزئية عظمى في النصف الأيسر،  $A[low..mid]$ . ولما كان واجباً أن تضم هذه الصفيقة  $A[mid]$ ، فإن حلقة **for** في السطور 3-7 يبدأ للمؤشر  $i$  عند  $mid$  ويناقص حتى القيمة  $low$ ، وبهذا تكون كل الصفيقات التي يدرسها هي من الشكل  $A[i..mid]$ . يعطي السطران 1-2 القيم البدائية للمتحوّلين:  $left-sum$  الذي يحوي أكبر مجموع تحقق حتى الآن، و  $sum$  الذي يحوي العناصر في  $A[i..mid]$ . وكلما وجدنا، في السطر 5، صفيقة جزئية  $A[i..mid]$  مجموعها أكبر من  $left-sum$ ، نعدّل  $left-sum$  إلى مجموع هذه الصفيقة الجزئية في السطر 6، ونعدّل في السطر 7 لمتحوّل  $max-left$  ليحتل قيمة المؤشر  $i$ . نعمل السطور 8-14 بطريقة ماثلة على النصف الأيمن  $A[mid + 1..high]$ . وهنا، يبدأ المؤشر  $j$  في حلقة **for** في السطور 10-14 من القيمة  $mid + 1$  ويزداد حتى  $high$ ، بحيث تكون كل صفيقة يدرسها من الشكل  $A[mid + 1..j]$ . وأخيراً، يعيد السطر 15 للمؤشرين  $max-left$  و  $max-right$  اللذين يحددان الصفيقة الجزئية التي تحتوي نقطة للتصنيف، إضافة إلى المجموع  $left-sum + right-sum$  وهو مجموع القيم في الصفيقة الجزئية  $A[max-left..max-right]$ .

إذا كانت الصفيقة الجزئية  $A[low..high]$  تحوي  $n$  عنصراً (بحيث  $n = high - low + 1$ )، فإننا ندعي أن الاستدعاء  $FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)$  يستغرق زمناً  $\Theta(n)$ . ولما كان كل تكرار لكل من حلقتي **for** يستغرق زمناً  $\Theta(1)$ ، فيكفي أن نأخذ عدد التكرارات الكلي. نقوم حلقة **for** في السطور 3-7 بإجراء  $mid - low + 1$  تكراراً، وحلقة **for** في السطور 10-14 بـ  $high - mid + 1$  تكراراً، وبهذا يكون عدد التكرارات الكلي هو

$$(mid - low + 1) + (high - mid) = high - low + 1 \\ = n .$$

الآن، وقد حصلنا على الإجراء ذي الزمن الخطي  $FIND-MAX-CROSSING-SUBARRAY$ ، يمكننا أن نكتب شبه الرماز لخوارزمية فرق-تسد تحل مسألة الصفيقة الجزئية العظمى:

$FIND-MAXIMUM-SUBARRAY(A, low, high)$

- 1 **if**  $high == low$
- 2     **return**  $\{low, high, A[low]\}$      // base case: only one element
- 3 **else**  $mid = \lfloor (low + high) / 2 \rfloor$
- 4      $(left-low, left-high, left-sum) =$   
        $FIND-MAXIMUM-SUBARRAY(A, low, mid)$
- 5      $(right-low, right-high, right-sum) =$   
        $FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)$
- 6      $(cross-low, cross-high, cross-sum) =$   
        $FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)$
- 7     **if**  $left-sum \geq right-sum$  and  $left-sum \geq cross-sum$

```

8      return (left-low, left-high, left-sum)
9      elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10     return (right-low, right-high, right-sum)
11     else return (cross-low, cross-high, cross-sum)

```

سيجد الاستدعاء البدائي لـ  $\text{FIND-MAXIMUM-SUBARRAY}(A, 1, A.length)$  صيغة جزئية عظمى لـ  $A[1..n]$ .

يعد الإجراء القوّدي  $\text{FIND-MAXIMUM-SUBARRAY}$ ، شأنه شأن الإجراء  $\text{FIND-MAX-CROSSING-SUBARRAY}$ ، ثلاثيةً تضم مؤشرين يحدّان صيغة جزئية عظمى، إضافةً إلى مجموع القيم في هذه الصيغة الجزئية العظمى. يختار السطر 1 تحقّق الحالة الأساسية، حيث تكون الصيغة الجزئية مكوّنة في عنصر واحد فقط. وعندما تكون الصيغة الجزئية مكوّنة من عنصر واحد، فإن فيها صيغة جزئية واحدة، هي نفسها، وبذلك يعيد السطر 2 ثلاثيةً فيها مؤشري البداية والنهاية لعنصر واحد فقط، إضافةً إلى قيمته. تعالج السطور 3-11 الحالة القوّدية. يقوم السطر 3 بجزء التقسيم، فيحسب مؤشر نقطة المنتصف، وهو  $mid$ . نسقي الصيغة الجزئية  $A[low..mid]$  *الصيغة الجزئية اليسرى*  $left\ subarray$  والصيغة  $A[mid + 1..high]$  *الصيغة الجزئية اليمنى*  $right\ subarray$ . ولما كنا نعرف أن الصيغة الجزئية  $A[low..high]$  تضم على الأقل عنصرين فإن كل من الصيغتين الجزئيتين اليسرى واليمنى ستضمّنان على الأقل عنصرًا واحدًا. يسود السطران 4 و 5 على المسألة من خلال الاستدعاء القوّدي لإيجاد الصيغتين الجزئيتين العظميين في كلٍّ من الصيغة الجزئية اليسرى ثم اليمنى. تقوم السطور 6-11 بجزء التجميع. يكتشف السطر 6 صيغة جزئية عظمى بحيث تحوي نقطة المنتصف. (تذكّر أننا نعتبر أن السطر 6 جزءًا من مرحلة تجميع الحل، لأنه يحلّ مسألة جزئية لا تمثّل متسخًا أصغر من المسألة الأصلية.) فإذا دلّ اختبار السطر 7 على أن الصيغة الجزئية اليسرى تحتوي صيغة جزئية ذات أكبر مجموع، فإن السطر 8 يعيد هذه الصيغة الجزئية العظمى. وإلا فإن دلّ اختبار السطر 9 على أن الصيغة الجزئية اليمنى تحتوي صيغة جزئية ذات أكبر مجموع، فإن السطر 10 يعيد هذه الصيغة الجزئية العظمى. وإذا لم تحتِ الصيغة الجزئية اليسرى ولا اليمنى صيغة جزئية تحقق أكبر مجموع، فمن المؤكّد أن الصيغة الجزئية العظمى تضم نقطة المنتصف، ويعيدها السطر 11.

### تحليل خوارزمية فرق-تسمد

سنضع فيما يلي علاقةً عؤدية تصف زمن تنفيذ الإجراء القوّدي  $\text{FIND-MAXIMUM-SUBARRAY}$ . سنستخدم، كما فعلنا عندما حللنا الفرز بالدمج في المقطع 2.3.2، الفرضية المبسطة وهي أن حجم المسألة الأصلية هو من قوى 2، بحيث تكون أحجام كل للسائل الجزئية أعدادًا صحيحة. نرمز لزمن تنفيذ الإجراء  $\text{FIND-MAXIMUM-SUBARRAY}$  على صيغة جزئية مؤلفة من  $n$  عنصرًا بـ  $T(n)$ . ونفترض، للمتبدئين، أن السطر 1 يستغرق زمنًا ثابتًا. وتكون الحالة القاعدية، عندما  $n = 1$ ، سهلة: يستغرق السطر 2 زمنًا ثابتًا، إذن



$$T(1) = \Theta(1) . \quad (5.4)$$

نحدث الحالة القودية عندما  $n > 1$ . يستغرق السطران 1 و 3 زمنًا ثابتًا. كل من المسألتين الجزئيتين المحلولتين في السطرين 6 و 5 هي مسألة على صيغة جزئية من  $n/2$  عنصرًا (تضمن لنا فرضيتنا بأن حجم المسألة الأصلية من قوى 2 أن  $n/2$  هو عدد صحيح) ، وبهذا يستغرق حل كل منهما زمنًا  $T(n/2)$ . ولما كان علينا حل مسألتين جزئيتين - للصفيفتين الجزئيتين اليسرى واليمنى - فإن مساهمة السطرين 4 و 5 في زمن التنفيذ تصل إلى  $2T(n/2)$ . وكما رأينا قبل قليل، فإن استدعاء FIND-MAX-CROSSING-SUBARRAY في السطر 6 يستغرق زمنًا  $\Theta(n)$ . تستغرق السطور 7-11 زمنًا  $\Theta(1)$  فقط. وبهذا يكون لدينا في الحالة القودية:

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n) . \end{aligned} \quad (6.4)$$

وبدمج المعادلتين (5.4) و (6.4) نحصل على علاقة عودية لـ  $T(n)$  زمن تنفيذ الإجراء FIND-

MAXIMUM-SUBARRAY:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 . \end{cases} \quad (7.4)$$

إن هذه العلاقة العودية تماثل للعلاقة القودية في (1.4) للفرز بالدمج. وسنرى من الطريقة الرئيسة في المقطع 5.4 أن حل هذه العلاقة العودية هو  $T(n) = \Theta(n \lg n)$ . ويمكنك أيضًا أن تعاود النظر إلى شجرة العودية في الشكل 5.2 لتدرك لماذا ينبغي أن يكون الحل هو  $T(n) = \Theta(n \lg n)$ .

وهكذا نرى أن طريقة فرق-تسد تعطي بالمقارنة خوارزمية أسرع من الطريقة الفجة. فبعد أن عَرَفْنَا سابقًا طريقة الفرز بالدمج والآن عَرَفْنَا مسألة الصيغة الجزئية العظمى، بدأنا نكوِّن فكرة عن مدى قوة طريقة فرق-تسد. ستعطينا هذه الطريقة في بعض الأحيان أسرع الحلول لمسألة ما بالمقارنة، وفي أحيان أخرى قد يكون بمقدورنا تحقيق حلٍّ أفضل. وكما يبيِّن التحرين 5-1.4، هناك في الواقع خوارزمية ذات زمن خطي لمسألة الصيغة الجزئية العظمى، وهي لا تستخدم مبدأ فرق-تسد.

## تمارين

### 1-1.4

ما الذي يعيده إجراء FIND-MAXIMUM-SUBARRAY عندما تكون كل عناصر  $A$  سالبة؟

### 2-1.4

اكتب شبه رماز للطريقة الفجة في حل مسألة الصيغة الجزئية العظمى. يجب أن تُثَبِّتَ إجرائيتك في زمن  $\Theta(n^2)$ .

### 3-1.4

نَحْزُ كُلًّا من الخوارزمية ذات الطريقة الفجة والخوارزمية القودية لمسألة الصيغة الجزئية العظمى على حاسوبك.

ما هو حجم المسألة  $n_0$  الذي يمثل نقطة التحاوز التي تتفوق بدنا منها الخوارزمية القودية على خوارزمية الطريقة الفخة؟ ثم غير الحالة القاعدية للخوارزمية القودية لتستخدم الخوارزمية ذات الطريقة الفخة كلما كان حجم المسألة أقل من  $n_0$ . هل يغير هذا التعديل من نقطة التحاوز هذه؟

#### 4-1.4

افترض أننا نغير تعريف مسألة الصيغة الجزئية العظمى لتسمح للنتيجة بأن تكون صيغة جزئية فارغة، حيث إن مجموع قيم صيغة جزئية فارغة هو 0. كيف يمكنك أن تقرر أيًا من الخوارزمتين التي لا تسمح بصيغيات جزئية فارغة بأن تتبع أن تكون النتيجة صيغة جزئية فارغة.

#### 5-1.4

استخدم الأفكار التالية لنظور خوارزمية غير عودية، ذات زمن خطي لحل مسألة الصيغة الجزئية العظمى. ابدأ من النهاية اليسرى للصيغة وتقدم نحو اليمين، محتفظًا بالصيغة الجزئية العظمى التي وقعت عليها قبل وصولك إلى المرحلة التالية. بمعرفتك الصيغة الجزئية العظمى في  $A[1..j]$ ، عَمِّمِ الجواب للثور على الصيغة الجزئية العظمى التي تنتهي بالمؤشر  $j+1$  باستخدام الملاحظة التالية: إن صيغة جزئية عظمى من  $A[1..j+1]$  هي إما صيغة جزئية عظمى من  $A[1..j]$  وإما صيغة جزئية من  $A[i..j+1]$  لقيم مؤشر  $i$  ما  $1 \leq i \leq j+1$ . حدّد صيغة جزئية من الشكل  $A[i..j+1]$  في زمن ثابت اعتمادًا على معرفتك لصيغة جزئية عظمى تنتهي عند المؤشر  $j$ .

## 2.4 خوارزمية شتراسن لجداء المصفوفات

إذا كنت قد رأيت مصفوفات قبلًا، فربما أنت تعرف كيف تقوم بعملية جدائهما. (وإلا، فعليك أن تقرأ المقطع ت.1 في الملحق ث.). إذا كانت  $A = (a_{ij})$  و  $B = (b_{ij})$  مصفوفتين مربعيتين  $n \times n$ ، فإن جداءهما هو المصفوفة:  $C = A \cdot B$ ، ونعرّف العنصر  $c_{ij}$  حيث  $i, j = 1, 2, \dots, n$  بالعلاقة

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj} \quad (8.4)$$

يجب أن نحسب  $n^2$  عنصرًا في المصفوفة، كلٌّ منها هو مجموع  $n$  قيمة. يأخذ الإجراء التالي مصفوفتين  $n \times n$  هما  $A$  و  $B$  ويحسب جداءهما، معبّدًا مصفوفة الجداء  $C$  ذات البعدين  $n \times n$ . نفترض أن لكل مصفوفة واصفة rows تعطي عدد السطور فيها.

SQUARE-MATRIX-MULTIPLY( $A, B$ )

1  $n = A.rows$

2 let  $C$  be a new  $n \times n$  matrix

```

3 for i = 1 to n
4   for j = 1 to n
5     cij = 0
6     for k = 1 to n
7       cij = cij + aik . bkj
8 return C

```

يعمل الإجراء SQUARE-MATRIX-MULTIPLY كالتالي: نحسب حلقة **for** في السطور 3-7 عناصر كل سطر  $i$ . وفي كل سطر  $i$ ، نحسب الحلقة **for** في السطور 4-7 كل عنصر من العناصر  $c_{ij}$  لكل عمود  $j$ . يستبدل السطر 5 قيمة  $c_{ij}$  بالصفر قبل أن نبدأ بحساب المجموع المعطى في المعادلة (8.4)، ويضيف كل تكرار من الحلقة **for** في السطرين 6-7 حدًا جديدًا إلى المعادلة (8.4).

لما كانت كل من حلقات **for** الثلاث للتداعلة تتكرر  $n$  تكرارًا تمامًا، ولما كان كل تنفيذ للسطر 7 يستغرق زمانًا ثابتًا، فإن الإجراء SQUARE-MATRIX-MULTIPLY يستغرق زمانًا  $\Theta(n^3)$ .

قد تعتقد بادئ الأمر أن أية خوارزمية لجداء المصفوفات يجب أن تستغرق زمانًا  $\Omega(n^3)$ ، وذلك لأن التعريف الطبيعي لجداء المصفوفات يتطلب هذا العدد من عمليات الجداء. إلا أن اعتقادك هذا غير صحيح: إذ إن لدينا طريقة لحساب جداء المصفوفات في زمن  $O(n^3)$ . وسنرى في هذا المقطع خوارزمية شتراسن Strassen الفؤدية المتميزة لجداء مصفوفتين  $n \times n$ . تُنفَّذ هذه الخوارزمية في زمن  $\Theta(n^{2.81})$  سينيته في المقطع 5.4. ولما كانت قيمة  $\lg 7$  تقع بين 2.80 و 2.81، فإن خوارزمية شتراسن تُنفَّذ في زمن  $O(n^{2.81})$ ، وهذا أفضل مقارنة بالإجراء البسيط SQUARE-MATRIX-MULTIPLY.

#### خوارزمية فرق-تسد بسيطة

سنفترض، حتى نبقي الأمور بسيطة عندما نستخدم خوارزمية فرق-تسد divide-and-conquer لحساب مصفوفة الجداء  $C = A \cdot B$ ، أن  $n$  هي من قوى 2 الصحيحة في كل المصفوفات  $n \times n$ . نحن نتبنى هذه الفرضية لأننا في كل خطوة تقسيم: سنقسم المصفوفات  $n \times n$  إلى أربع مصفوفات  $n/2 \times n/2$ ، وبافتراض أن  $n$  من قوى 2 الصحيحة، نضمن أن البعد  $n/2$  هو عدد صحيح، مادام  $n \geq 2$ .

افترض أننا نقسم كلًا من المصفوفات  $A$  و  $B$  إلى أربع مصفوفات  $n/2 \times n/2$ .

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}, \quad (9.4)$$

وبهذا نعيد كتابة المعادلة  $C = A \cdot B$  بالشكل

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}. \quad (10.4)$$

توافق المعادلة (10.4) للمعادلات الأربع التالية:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \quad (11.4)$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \quad (12.4)$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \quad (13.4)$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \quad (14.4)$$

تحدد كل من هذه المعادلات الأربع جداءين لمصفوفات  $n/2 \times n/2$ ، ومجموع ناتجتي الجداء وهو مصفوفة  $n/2 \times n/2$ . يمكننا استخدام هذه المعادلات لإيجاد خوارزمية فرق-تسد مباشرة عؤدية:

#### SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)

```

1   $\pi = A.rows$ 
2  let C be a new  $\pi \times \pi$  matrix
3  if  $\pi == 1$ 
4       $C_{11} = a_{11} \cdot b_{11}$ 
5  else partition A, B, and C in equations (9.4)
6       $C_{11} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{11})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{21})$ 
7       $C_{12} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{11}, B_{12})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{12}, B_{22})$ 
8       $C_{21} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{11})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{21})$ 
9       $C_{22} = \text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{21}, B_{12})$ 
           +  $\text{SQUARE-MATRIX-MULTIPLY-RECURSIVE}(A_{22}, B_{22})$ 
10 return C
```

إن شبه الرمز السابق يحفي تفصيل تنحيز دقيق وحيد، لكنه تفصيل هام. كيف تقسم المصفوفات في السطر 9؟ فلو كان علينا إنشاء 12 مصفوفة جديدة  $n/2 \times n/2$  لكننا أمضينا زمناً  $\Theta(n^2)$  في نسخ العناصر. في الواقع، يمكننا تقسيم المصفوفات دون نسخ العناصر، والسر يكمن في الاعتماد على حساب المؤشرات. نحدد مصفوفة جزئية بتحديد مجال من مؤشرات السطور ومجال من مؤشرات الأعمدة في المصفوفة الأصلية. أي إننا نحصل في النهاية على تمثيل للمصفوفة الجزئية مختلف قليلاً عن طريقة تمثيل المصفوفة الأصلية، وهذا هو التفصيل الدقيق الذي نفعله في شبه الرمز. وفائدة ذلك هي أن تنفيذ السطر 5 لا يستغرق إلا زمناً  $\Theta(1)$ ، إذ إننا نحدد المصفوفات الجزئية من خلال حساب المؤشرات (ولكننا نسرى أنه سواء أقمنا بالنسخ أم بالتقسيم في المكان، فإن ذلك لا يؤثر بالمقارنة على زمن التنفيذ الكلي).

نشقي الآن علاقة عؤدية توصف زمن تنفيذ SQUARE-MATRIX-MULTIPLY-RECURSIVE. ليكن  $T(n)$  الزمن اللازم لحساب جداء مصفوفتين  $n \times n$  باستخدام هذا الإجراء. في الحالة القاعدية، عندما  $n = 1$ ، نقوم فقط بعملية جداء سُلمِي في السطر 4 وبهذا يكون

$$T(1) = \Theta(1) \quad (15.4)$$

تحدث الحالة القودية عندما يكون  $n > 1$ . وكما ناقشنا سابقاً، فإن تقسيم المصفوفات في السطر 5 يستغرق زمناً  $\Theta(1)$  باستخدام حساب المؤشرات. نقوم في السطور 6-9 باستدعاء SQUARE-MATRIX-MULTIPLY-RECURSIVE عُودياً 8 مرات. ولما كان كل استدعاء يحسب جداء مصفوفتين  $n/2 \times n/2$ ، فإن مساهمته في زمن التنفيذ الكلي  $T(n/2)$ ، ويكون الزمن الكلي الذي تستغرقه الاستدعاءات القودية الثمانية هو  $8T(n/2)$ . ويجب أن نأخذ أيضاً في الحسبان عمليات جمع المصفوفات الأربع في السطور 6-9، فكلٌّ من هذه المصفوفات فيها  $n^2/4$  عنصرًا، وبذلك فإن كلاً من عمليات جمع المصفوفات الأربع تستغرق زمناً  $\Theta(n^2)$ . ولما كان عدد مرات جمع المصفوفات ثانياً، فإن الزمن الكلي الذي يستغرقه جمع المصفوفات في السطور 6-9 هو  $\Theta(n^2)$ . (نستخدم هنا أيضاً حساب المؤشرات لنضع نتائج جمع المصفوفات في مواقعها الصحيحة داخل المصفوفة C، بزمٍ إضافي  $\Theta(1)$  لكل عنصر.) إذن الزمن في الحالة القودية هو مجموع زمن التقسيم وزمن جميع الاستدعاءات القودية، وزمن جمع المصفوفات الناتجة عن الاستدعاءات القودية:

$$\begin{aligned} T(n) &= \Theta(1) + 8T(n/2) + \Theta(n^2) \\ &= 8T(n/2) + \Theta(n^2). \end{aligned} \quad (16.4)$$

لاحظ أنه إذا تخَّزنا التقسيم بنسخ المصفوفات، لاستغرق ذلك زمناً  $\Theta(n^2)$ ، ولما تغيَّرت العلاقة القودية، ولازداد زمن التنفيذ الكلي بمعامل ثابت فقط.

إن تجميع المعادلتين (15.4) و (16.4) يعطينا العلاقة لزمن تنفيذ SQUARE-MATRIX-MULTIPLY-RECURSIVE:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 8T(n/2) + \Theta(n^2) & \text{if } n > 1 \end{cases} \quad (17.4)$$

سنرى باستخدام الطريقة الرئيسة في المقطع 5.4 أن حل العلاقة القودية (17.4) هو  $T(n) = \Theta(n^3)$ . أي إن طريقة فرق-تد البسيطة هذه ليست أسرع من إجراء SQUARE-MATRIX-MULTIPLY المباشر.

قبل أن نتابع دراسة خوارزمية شتراسن، سننظر من أين جاءت مركبات المعادلة (16.4). إن تقسيم كل مصفوفة  $n \times n$  بحساب المؤشرات يستغرق زمناً  $\Theta(1)$ ، إلا أنه لدينا مصفوفتين يجب تقسيمهما، وعلى الرغم من أنه بإمكانك القول أن تقسيم مصفوفتين يستغرق  $\Theta(2)$ ، إلا أن الثابت 2 متضمن في التدوين- $\Theta$ . إن جمع مصفوفتين، في كلٍّ منهما  $k$  عنصرًا، يستغرق زمناً  $\Theta(k)$ . ولما كانت المصفوفات التي نجمعها ذات  $n^2/4$  عنصرًا، يمكننا أن نقول إن جمع كل زوج من المصفوفات يستغرق زمناً  $\Theta(n^2/4)$ ، وهنا أيضاً يتضمن التدوين- $\Theta$  وجود المعامل الثابت  $1/4$ ، فنقول إن جمع مصفوفتين  $n/2 \times n/2$  يستغرق زمناً  $\Theta(n^2)$ . لدينا أربع عمليات جمع مثل هذه المصفوفات، وهنا أيضاً عوضاً عن أن نقول إن ذلك يستغرق  $\Theta(4n^2)$ ، فإننا نكتفي بالقول إننا تستغرق زمناً  $\Theta(n^2)$ . (طبعا، قد تلاحظ أنه كان بإمكاننا القول إن جمع المصفوفات الأربع يستغرق زمناً  $\Theta(4n^2/4)$ ، وإن  $4n^2/4 = n^2$ ، إلا أن الفكرة هنا أن التدوين- $\Theta$  يتضمن المعاملات

الثانية، مهما كانت قيمتها.) وهكذا تنتهي مع حلين  $\Theta(n^2)$  و  $\Theta(1)$  يمكن تجميعهما في حد واحد. أما عندما نحسب الاستدعاءات القودية الثمانية، فليس بإمكاننا أن نعتبر للمعامل الثابت 8 متضمنًا. بمعنى آخر، يجب أن نقول بوضوح إننا تستغرق مًا  $8T(n/2)$  بدلاً من أن نكتفي بالقول  $T(n/2)$ . يمكنك أن تستشف ذلك بمعاودة النظر إلى شجرة القودية في الشكل 5.2، للعلاقة القودية (1.2) (وهي عمائل العلاقة القودية (7.4))، مع الحالة القودية  $T(n) = 2T(n/2) + \Theta(n)$ . حدد للمعامل 2 عدد الأبناء لكل عقدة في الشجرة، وهذا حدد بدوره عدد الحدود التي تدخل في المجموع في كل مستوى من الشجرة. لو كان لنا أن نتجاهل للمعامل 8 في المعادلة (16.4) أو للمعامل 2 في العلاقة القودية (1.4) لأصبحت شجرة القودية خطية فقط عوضاً عن أن تكون "كثيفة"، ولاخضرت مساهمة كل مستوى في المجموع في حد واحد فقط. تذكر أنه على الرغم من أن التدوين المقارب يتضمن جداء للمعاملات الثابتة، إلا أن التدوين القودي مثل  $T(n/2)$  لا يتضمنها أبداً.

### طريقة شتراسن

الفكرة الأساسية في طريقة شتراسن Strassen's method هي تقليل كثافة شجرة القودية قليلاً ففرضاً عن إجراء ثمانية جداءات قودية لمصفوفات  $n/2 \times n/2$  بحرى سبعة فقط، ومقابل إلغاء جداء مصفوفات واحد، يجري المزيد من جمع المصفوفات، ولكن فقط عدد ثابت منها. وكما ذكرنا سابقاً، فإن عددًا ثابتاً من جمع المصفوفات سيكون متضمناً في تدوين  $\Theta$  عندما نبني العلاقة القودية التي توصف زمن التنفيذ. إن طريقة شتراسن ليست بدئية بثنائياً (قد يكون هذا التصريح هو الأخطر في هذا الكتاب) وهي مكونة من 4 خطوات:

1. نقسم مصفوفات الدخل الدخل  $A$  و  $B$  مصفوفة المخرج  $C$  إلى مصفوفات جزئية  $n/2 \times n/2$  كما في المعادلة (9.4). تستغرق هذه الخطوة زمناً  $\Theta(1)$  بحساب للإشارات غاملاً كما في الإجراء SQUARE-MATRIX-MULTIPLY.
  2. ننشئ 10 مصفوفات  $S_1, S_2, \dots, S_{10}$  كل منها  $n/2 \times n/2$ ، وهي ناتج جمع أو طرح مصفوفتين من بين تلك المنشأة في الخطوة 1. يمكن إنشاء هذه المصفوفات العشر كلها في زمن  $\Theta(n^2)$ .
  3. باستخدام المصفوفات الجزئية المنشأة في الخطوة 1، والمصفوفات العشر المنشأة في الخطوة 2، نحسب عؤدياً 7 جداءات مصفوفات  $P_1, P_2, \dots, P_7$ ، كل مصفوفة  $P_i$  هي  $n/2 \times n/2$ .
  4. نحسب المصفوفات الجزئية المطلوبة  $C_{11}, C_{12}, C_{21}, C_{22}$  التي تكون المصفوفة  $C$  بجمع أو بطرح تركيبات مختلفة من المصفوفات  $P_i$ ، يمكن حساب هذه المصفوفات الأربع جميعاً في زمن  $\Theta(n^2)$ .
- سنرى تفاصيل الخطوات 2-4 بعد قليل، ولكن لدينا الآن ما يكفي من المعلومات لبنى العلاقة القودية

التي تحكم زمن تنفيذ شتراسن. وقد افترضنا أنه ما إن يصل حجم المصفوفة  $n$  إلى 1، حتى نقوم بجداء سلمي بسيط، تمامًا كما في السطر 4 من SQUARE-MATRIX MULTIPLY-RECURSIVE. عندما يكون  $n > 1$ ، تستغرق الخطوات 1 و 2 و 4 زمنًا مجمله  $\Theta(n^2)$ . وتتطلب الخطوة 3 إجراء سبع جداءات للمصفوفات  $n/2 \times n/2$ . وبهذا، نحصل على العلاقة التّؤدّية التالية لزمن خوارزمية شتراسن:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 7T(n/2) + \Theta(n^2) & \text{if } n > 1. \end{cases} \quad (18.4)$$

لقد قابضنا جداءً مصفوفةً واحدًا مقابل عددٍ ثابتٍ من عمليات جمع المصفوفات. وسنرى لاحقًا عندما نتمعّن في معرفة العلاقات التّؤدّية وحلولها أن هذه للتّفاضة تؤدي إلى زمن تنفيذ مقارب أدنى. واعتمادًا على الطريقة الرئيسية في المقطع 5.4، نجد أن حل العلاقة التّؤدّية (18.4) هو  $T(n) = \Theta(n^{\lg 7})$ . وفيما يلي تفصيل ذلك. ننشئ في الخطوة 2 المصفوفات العشر التالية:

$$\begin{aligned} S_1 &= B_{12} - B_{22}, \\ S_2 &= A_{11} + A_{12}, \\ S_3 &= A_{21} + A_{22}, \\ S_4 &= B_{21} - B_{11}, \\ S_5 &= A_{11} + A_{22}, \\ S_6 &= B_{11} + B_{22}, \\ S_7 &= A_{12} - A_{22}, \\ S_8 &= B_{21} + B_{22}, \\ S_9 &= A_{11} - A_{21}, \\ S_{10} &= B_{11} + B_{12}. \end{aligned}$$

ولما كان علينا أن نجمع أو نطرح مصفوفات  $n/2 \times n/2$  عشر مرات، فإن هذه الخطوة تستغرق في الحقيقة زمنًا  $\Theta(n^2)$ .

بُحري، في الخطوة 3، سبع جداءات عُدّتها لمصفوفات  $n/2 \times n/2$  لحساب المصفوفات التالية، التي ينتج كل منها عن مجموع أو فرق جداءات لمصفوفات جزئية من  $A$  و  $B$ :

$$\begin{aligned} P_1 &= A_{11} \cdot S_1 = A_{11} \cdot B_{12} - A_{11} \cdot B_{22}, \\ P_2 &= S_2 \cdot B_{22} = A_{11} \cdot B_{22} + A_{12} \cdot B_{22}, \\ P_3 &= S_3 \cdot B_{11} = A_{21} \cdot B_{11} + A_{22} \cdot B_{11}, \\ P_4 &= A_{22} \cdot S_4 = A_{22} \cdot B_{21} - A_{22} \cdot B_{11}, \\ P_5 &= S_5 \cdot S_6 = A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22}, \\ P_6 &= S_7 \cdot S_8 = A_{12} \cdot B_{21} + A_{12} \cdot B_{22} - A_{22} \cdot B_{21} - A_{22} \cdot B_{22}, \\ P_7 &= S_9 \cdot S_{10} = A_{11} \cdot B_{11} + A_{11} \cdot B_{12} - A_{21} \cdot B_{11} - A_{21} \cdot B_{12}. \end{aligned}$$

لاحظ أن عمليات الجداء الوحيدة التي نحتاج إلى إنجازها هي العمليات الموجودة في العمود الأوسط من المعادلات السابقة. أما العمود الأيمن فهو مجرد أن يبيّن ماذا تساوي هذه الجداءات، بدلالة المصفوفات الجزئية الأصلية المنشأة في الخطوة 1.

تقوم الخطوة 4 بجمع وطرح المصفوفات  $P_i$  المحسوبة في الخطوة 3 لتبني المصفوفات الجزئية  $n/2 \times n/2$  الأربع التي تكوّن مصفوفة الجداء  $C$ . نبدأ بالمصفوفة

$$C_{11} = P_5 + P_4 - P_2 + P_8 .$$

وينشر الطرف الأيمن، بتعميم كل  $P_i$  بسطها الخاص وبوضع الحدود التي يحدف أحدها الآخر في العمود نفسه، نجد أن  $C_{11}$  تساوي:

$$\begin{array}{r} A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} - A_{12} \cdot B_{22} \\ - A_{11} \cdot B_{22} - A_{22} \cdot B_{22} - A_{22} \cdot B_{21} + A_{12} \cdot B_{22} - A_{12} \cdot B_{21} \\ \hline A_{11} \cdot B_{11} + A_{12} \cdot B_{21} , \end{array}$$

وهذا يتوافق مع المعادلة (11.4).

بالمثل نضع  $C_{12} = P_2 + P_2$  ، فنجد أن  $C_{12}$  تساوي:

$$\begin{array}{r} A_{11} \cdot B_{12} - A_{11} \cdot B_{22} \\ + A_{11} \cdot B_{22} + A_{12} \cdot B_{22} \\ \hline A_{11} \cdot B_{12} + A_{12} \cdot B_{22} , \end{array}$$

وهذا يتوافق مع المعادلة (12.4).

وبوضع  $C_{21} = P_3 + P_4$  ، نجد أن  $C_{21}$  تساوي:

$$\begin{array}{r} A_{21} \cdot B_{11} + A_{22} \cdot B_{11} \\ - A_{22} \cdot B_{11} + A_{22} \cdot B_{21} \\ \hline A_{21} \cdot B_{11} + A_{22} \cdot B_{21} , \end{array}$$

وهذا يتوافق مع المعادلة (13.4).

وأخيراً، نضع  $C_{22} = P_5 + P_1 - P_3 - P_7$  ، فنجد أن  $C_{22}$  تساوي:

$$\begin{array}{r} A_{11} \cdot B_{11} + A_{11} \cdot B_{22} + A_{22} \cdot B_{11} + A_{22} \cdot B_{22} \\ - A_{11} \cdot B_{22} + A_{11} \cdot B_{12} - A_{22} \cdot B_{11} - A_{21} \cdot B_{11} \\ - A_{11} \cdot B_{11} - A_{11} \cdot B_{12} + A_{21} \cdot B_{11} + A_{21} \cdot B_{12} \\ \hline A_{22} \cdot B_{22} + A_{21} \cdot B_{12} , \end{array}$$

وهذا يتوافق مع المعادلة (14.4). وبالمجمل، فإننا نقوم في الخطوة 4 بجمع وطرح مصفوفات  $n/2 \times n/2$  ثماني مرات، وبذلك تستغرق هذه الخطوة قطعاً زمناً  $\Theta(n^2)$ .



وهكذا نرى أن خوارزمية شتراسن، للكونة من الخطوات 1-4، تعطي جداء المصفوفات الصحيح، وأن العلاقة القودية (18.4) توصف زمن تنفيذها. ولما كان حل هذه العلاقة القودية هو  $T(n) = \Theta(n^4)$ ، كما سنرى في اللقطع 5.4، فإن طريقة شتراسن أسرع بالمقارنة من إجراء الجداء المباشر SQUARE-MATRIX-MULTIPLY. نتاقش للملاحظات المذكورة في نهاية الفصل بعض الجوانب العملية لخوارزمية شتراسن.

### تعارين

ملاحظة: على الرغم من أن التعارين 2.4-3، و 2.4-4، و 2.4-5 تتعلق بخوارزميات معدلة عن خوارزمية شتراسن، فمن المستحسن قراءة اللقطع 5.4 قبل أن نحاول حلها.

#### 1-2.4

استخدم خوارزمية شتراسن لحساب جداء المصفوفتين

$$\begin{pmatrix} 1 & 3 \\ 7 & 5 \end{pmatrix} \begin{pmatrix} 6 & 8 \\ 4 & 2 \end{pmatrix}.$$

مبيناً تفاصيل عملك.

#### 2-2.4

اكتب شبه رماز لخوارزمية شتراسن.

#### 3-2.4

كيف يمكنك تغيير خوارزمية شتراسن لحساب جداء مصفوفتين  $n \times n$  حيث لا تكون  $n$  من قوى 2 الصحيحة؟ بَيِّن أن الخوارزمية الناتجة تُنفَّذ في زمن  $\Theta(n^4)$ .

#### 4-2.4

ما هو أكبر عدد  $k$  بحيث، إذا كان بإمكاننا إجراء جداء مصفوفتين  $3 \times 3$  باستخدام  $k$  جداء (دون الاستفادة من تبادلية الجداء)، يكون بإمكاننا إجراء جداء المصفوفات  $n \times n$  في زمن  $\Theta(n^k)$ ؟ كيف يكون زمن تنفيذ هذه الخوارزمية؟

#### 5-2.4

اكتشف V. Pan طريقة لجداء مصفوفتين  $68 \times 68$  باستخدام 132,464 جداء، وطريقة لجداء مصفوفتين  $70 \times 70$  باستخدام 143,640 جداء، وطريقة لجداء مصفوفتين  $72 \times 72$  باستخدام 155,424 جداء. أي الطرق تعطي أفضل زمن تنفيذ مقارب عند استخدامها في خوارزمية فرق-تسد لجداء المصفوفات؟ كيف تتقارن هذه الخوارزمية بخوارزمية شتراسن؟

#### 6-2.4

ما سرعة تنفيذ جداء مصفوفة  $n \times kn$  بمصفوفة  $n \times kn$  باستخدام خوارزمية ستراشن إجراءً فرعيًا؟ أحب عن السؤال نفسه بقلب ترتيب مصفوفتي الدخل.

7-2.4

يُبين كيف تُجرى عملية جداء العددين العقديين  $a + bi$  و  $c + di$  باستخدام ثلاث عمليات جداء أعداد حقيقية فقط. ينبغي أن يكون دخل الخوارزمية الأعداد  $a$ ،  $b$ ،  $c$  و  $d$ ، وأن تعيد المركبة الحقيقية  $ac - bd$  والمركبة التخيلية  $ad + bc$  كلاً منهما على حدة.

### طريقة التعويض لحل العلاقات القُوْدِيَّة

3.4

بعد أن رأينا كيف توصف العلاقات القُوْدِيَّة لزمن تنفيذ خوارزميات فرق-تسد، سنتعلم كيف نحل هذه العلاقات القُوْدِيَّة. نبدأ في هذا المقطع بطريقة "التعويض".

تتضمن طريقة التعويض *substitution method* المستخدمة في حل العلاقات القُوْدِيَّة خطوتين:

1. تخمين أسلوب الحل.

2. استخدام الاستقراء الرياضي للثبوت، وبيان صحة الحل.

يعود اسم الطريقة إلى أننا نعوّض الحلّ للتحقق للدالة عندما نطبق فرضية الاستقراء على قيم أصغر. هذه الطريقة فعالة، إلا أنه يجب أن نكون قادرين على تخمين شكل الجواب حتى تتمكن من استخدامها.

يمكن استخدام طريقة التعويض لإيجاد حدود عليا أو دنيا للعلاقة القُوْدِيَّة. وكمثال على ذلك، نحدّد حدًا أعلى للعلاقة القُوْدِيَّة

$$T(n) = 2T(\lfloor n/2 \rfloor) + n, \quad (19.4)$$

التي تشبه العلاقات القُوْدِيَّة (3.4) و (4.4). نغمن أن الحل هو  $T(n) = O(n \lg n)$ . تتطلب طريقة التعويض أن نبرهن أن  $T(n) \leq cn \lg n$  عند اختيار مناسب للثابت  $c > 0$ . نبدأ بافتراض أن هذا الحد محقق عند كل الأعداد الموجبة  $m < n$ ، وخاصة عند  $m = \lfloor n/2 \rfloor$ ، وهذا يعطي  $T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$ . وبالتعويض داخل العلاقة القُوْدِيَّة نحصل على

$$\begin{aligned} T(n) &\leq 2(c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\ &\leq cn \lg(n/2) + n \\ &\leq cn \lg n - cn \lg 2 + n \\ &\leq cn \lg n - cn + n \\ &\leq cn \lg n, \end{aligned}$$

حيث تكون الخطوة الأخيرة محققة مادام  $c \geq 1$ .

يفرض الاستقراء الرياضي علينا الآن أن نبين أن حلنا محقق عند الشروط الحدودية. وعادة ما نقوم بذلك بأن نبين أن الشروط الحدّية مناسبة باعتبارها حالات أساسية بالنسبة للبرهان بالاستقراء. إذن، يجب أن نبين

في حالة العلاقة القُودِيَّة (19.4) أنه بإمكاننا اختيار الثابت  $c$  كبيراً كفاية بحيث يكون الحد  $T(n) \leq cn \lg n$  محققاً أيضاً في حالة الشروط الحديثة. وقد يؤدي هذا للطلب في بعض الأحيان إلى بعض المشكلات. لنفترض، من أجل المناقشة، أن  $T(1) = 1$  هو الشرط الحُدِّي الوحيد للعلاقة القُودِيَّة. فإذا كان  $n = 1$ ، فإن الحد  $T(n) \leq cn \lg n$  يعطي المتراجحة  $0 \leq c1 \lg 1 = 0$ ، وهذا يتناقض  $T(1) = 1$ . وبناء على ذلك، فإن الحالة الأساسية لبرهاننا بالاستقراء غير محققة.

يمكننا تجاوز هذه الصعوبة في برهان فرضية الاستقراء على شرط حُدِّي محدّد ببذل القليل من الجهود الإضافي. ففي العلاقة القُودِيَّة (19.4) مثلاً، نستفيد من أن التدوين المقارب لا يتطلب سوى أن نبرهن أن  $T(n) < cn \lg n$  عندما  $n \geq n_0$ ، حيث  $n_0$  ثابت نختاره. نحفظ بالشرط الحُدِّي الذي يسبب لنا المشاكل  $T(1) = 1$ ، إلا أننا لا نأخذ في الحسبان في البرهان بالاستقراء. نقوم بذلك بأن نلاحظ أنه عندما  $n > 3$ ، لا تعتمد العلاقة القُودِيَّة مباشرة على  $T(1)$ . وهكذا، يمكننا استبدال  $T(2)$  و  $T(3)$  بـ  $T(1)$  باعتبارها حالات أساسية في برهاننا بالاستقراء، ونضع  $n_0 = 2$ . لاحظ أننا نغيّر بين الحالة الأساسية للعلاقة القُودِيَّة ( $n = 1$ )، والحالتين الأساسيتين للبرهان بالاستقراء ( $n = 2$  و  $n = 3$ ). باستخدام  $T(1) = 1$ ، نستنتج من العلاقة القُودِيَّة أن  $T(2) = 4$  و  $T(3) = 5$ . ويمكن الآن استكمال البرهان بالاستقراء للعلاقة  $T(n) \leq cn \lg n$  لثابت ما  $c \geq 1$ ، بأن نختار  $c$  كبيرة كفاية بحيث يكون  $T(2) \leq c2 \lg 2$  و  $T(3) \leq c3 \lg 3$ . وكما يبدو لنا، فإن أيّ اختيار لـ  $c \geq 2$  كافٍ للحالتين الأساسيتين  $n = 2$  و  $n = 3$ . سيكون من السهل توسيع الشروط الحديثة لمعظم العلاقات القُودِيَّة التي سندرجها لجعل فرضية الاستقراء سارية عندما تكون قيم  $n$  صغيرة، ولن نوضّح هذا النوع من التفاصيل دائماً.

#### صياغة تخمين جيد

لا يوجد، لسوء الحظ، طريقة عامة لتخمين الحلول الصحيحة للمعادلات القُودِيَّة. يتطلب تخمين الحل خبرة، وأحياناً بعض الإبداع. ولكن، لحسن الحظ، هناك بعض الطرق الكسبية heuristics التي يمكن أن تساعدك لتصبح تخميناً جيداً. يمكنك أيضاً استخدام شجرات القُودِيَّة، التي ستراها في المقطع 4.4 لتوليد تخمينات جيدة.

إذا كانت العلاقة القُودِيَّة مشابهة لعلاقة صادفتك من قبل، فإن تخمين حلّ مشابه سيكون معقولاً. مثال:

لنأخذ العلاقة القُودِيَّة

$$T(n) = 2T(\lfloor n/2 \rfloor + 17) + n,$$

التي تبدو صعبة بسبب القيمة "17" المضافة في محدد  $T$  في الطرف الأيمن. إلا أننا بالحدس نرى أن هذا الحد الإضافي غير قادر على التأثير تأثيراً جوهرياً على حل العلاقة القُودِيَّة. عندما تكون  $n$  كبيرة، لن يكون القارق بين  $\lfloor n/2 \rfloor$  و  $\lfloor n/2 \rfloor + 17$  كبيراً: كلاهما يقسم  $n$  إلى قسمين متعادلين تقريباً. ومن ثمّ، نغمن أن الحل هو

$T(n) = O(n \lg n)$ ، وهذا ما يمكنك التحقق منه باستخدام طريقة التعويض (انظر التمرين 6-3.4).

هناك طريقة أخرى للحصول على تخمين جيد تتمثل في برهان حدود عليا ودنيا غير ملاصقة للعلاقة القودية، ثم السعي لتقليل مجال البرية. مثلاً، قد نبدأ بحد أدنى للعلاقة القودية (19.4) من الشكل  $T(n) = \Omega(n)$ ، وذلك لأن العلاقة القودية فيها الحد  $n$ ، ويمكننا البرهان على حد أعلى مبدئي هو  $T(n) = O(n^2)$ . بعد ذلك، نقوم تدريجياً بتخفيض الحد الأعلى ورفع الحد الأدنى حتى نتقارب إلى الحل الصحيح الملائق بالمقاربة، وهو  $T(n) = \Theta(n \lg n)$ .

### حالات تتطلب دقة

في بعض الأحيان، يكون بإمكانك تخمين حد مقارب صحيح لحل العلاقة القودية، إلا أنه يبدو أن العمليات الرياضية لا تسير على ما يرام في برهان الاستقراء. وعادة ما تكمن للمشكلة في أن فرضية الاستقراء ليست بالقوة الكافية لبرهان الحد بتفاصيله. وعندما تواجه مثل هذه العقبة، كثيراً ما تسمح مراجعة التخمين عن طريق طرح حد من مرتبة أدنى من الحل للتخمين بأن تسير العمليات الرياضية كما يجب. لندرس العلاقة القودية

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1.$$

نخمن أن الحل هو  $T(n) = O(n)$ ، ونحاول أن نبين أن  $T(n) \leq cn$  مع اختيار مناسب لـ  $c$ . بتعويض تخميننا في العلاقة القودية، نحصل على

$$\begin{aligned} T(n) &\leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 \\ &= cn + 1 \end{aligned}$$

وهذا لا يقتضي أن  $T(n) \leq cn$  مهما كان اختيار  $c$ . وقد نرغب في محاولة حل تخمين أكبر، ولكن  $T(n) = O(n^2)$ . ومع أنه يمكننا التحقق من هذا التخمين الأكبر، فإن تخميننا الأصلي أن الحل هو  $T(n) = O(n)$  صحيح. وحتى نبين ذلك، علينا في الواقع أن نضع فرضية استقراء أقوى.

نرى بالجلس أن تخميننا صحيح تقريباً: فنحن بعيدون عنه بمقدار الثابت 1، وهو حد من درجة أصغر. إلا أن الاستقراء الرياضي لا يثبت حتى نبرهن الصيغة الدقيقة لفرضية الاستقراء. سنتجاوز هذه الصعوبة بأن نطرح من تخميننا السابق حداً من مرتبة أصغر، فيصبح تخميننا الجديد  $T(n) \leq cn - d$  حيث  $d \geq 0$  ثابت. لدينا الآن

$$\begin{aligned} T(n) &\leq (c\lfloor n/2 \rfloor - d) + (c\lceil n/2 \rceil - d) + 1 \\ &= cn - 2d + 1 \\ &\leq cn - d, \end{aligned}$$

مادام  $d \geq 1$ . ومثلما سبق، يجب اختيار الثابت  $c$  كبيراً كفاية ليحقق الشروط الجديدة.

قد تجد فكرة طرح حد من مرتبة أصغر غير بديهية. ولكن في النهاية، إذا كانت العمليات الرياضية لا تتم كما يجب، ألا يجدر بنا أن نزيد تخميننا قليلاً؟ ليس بالضرورة! عندما نستخدم الاستقراء الرياضي لبرهان حد أعلى، قد يكون في الواقع برهان حد أضعف أكثر صعوبة، لأننا نحتاج عند برهان هذا الحد الأضعف إلى استخدام الحد الأضعف نفسه غُودِيًا في البرهان: في مثالنا الحالي، وعندما تتضمن العلاقة التَّوَدِيَّة أكثر من حد غُودِي واحد، قمنا بطرح الحد ذي المرتبة الدنيا من الحد المقترح مرة لكل حد غُودِي. ففي المثال السابق، طرحنا الثابت  $d$  مرتين: مرة للحد  $T(\lfloor n/2 \rfloor)$ ، ومرة للحد  $T(\lfloor n/2 \rfloor)$ . فانتهى بنا الأمر إلى المتراجحة  $T(n) \leq cn - 2d + 1$ ، وكان من السهل إيجاد قيم  $d$  تجعل  $cn - 2d + 1$  أصغر من  $cn - d$  أو تساويه.

### تجاشي الثغرات

من السهل الوقوع في الخطأ عند استخدام التدوين للمقارب. بإمكاننا مثلاً، في العلاقة التَّوَدِيَّة (19.4)، أن نبرهن خطأً أن  $T(n) = O(n)$  بأن نحسن  $T(n) \leq$  وأن نناقش

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor) + n \\ &\leq cn + 1 \\ &= O(n), \quad \text{خطأ !!} \end{aligned}$$

إذ إن  $c$  ثابت. والخطأ هو أننا لم نبرهن الصيغة الدقيقة لفرضية الاستقراء، وهي  $T(n) \leq cn$ . إذن، سنبرهن صراحةً أن  $T(n) \leq c$  عندما نريد أن نبين أن  $T(n) = O(n)$ .

### تغيير المتحولات

في بعض الأحيان، قد تساعد عملية جبرية صغيرة على أن تجعل علاقة غُودِيَّة غير معروفة مشابهةً لأخرى رأيناها قبلاً. مثلاً، لتكن لدينا العلاقة التَّوَدِيَّة

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n,$$

التي تبدو صعبة. إلا أنه يمكننا أن نبسط هذه العلاقة التَّوَدِيَّة باستخدام تغيير المتحولات. وللتبسيط، لن نحتم بأمر تدوير القيم مثل  $\sqrt{n}$  لتكون طبيعية. وبإعادة تسمية  $m = \lg n$  نحصل على

$$T(2^m) = 2T(2^{m/2}) + m.$$

يمكننا الآن إعادة تسمية  $S(m) = T(2^m)$  لنحصل على العلاقة التَّوَدِيَّة الجديدة

$$S(m) = 2S(m/2) + m,$$

التي تشابه كثيراً العلاقة (19.4). وفي الحقيقة فإن العلاقة التكرارية الجديدة لها الحل نفسه

$S(m) = O(m \lg m)$ . وبالقوة من  $S(m)$  إلى  $T(n)$ ، نحصل على  
 $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \lg \lg n)$ .

تمارين

1-3.4

بيّن أن حل العلاقة  $T(n) = T(n-1) + n$  هو  $O(n^2)$ .

2-3.4

بيّن أن حل العلاقة  $T(n) = T(\lfloor n/2 \rfloor) + 1$  هو  $O(\lg n)$ .

3-3.4

رأينا أن حل  $T(n) = 2T(\lfloor n/2 \rfloor) + n$  هو  $O(n \lg n)$ . بيّن أن حل هذه العلاقة القودية هو أيضاً  $\Omega(n \lg n)$ . استنتج أن الحل هو  $\Theta(n \lg n)$ .

4-3.4

بيّن أنه يمكننا بتغيير فرضية الاستقراء، أن نتجاوز المشكلة المتعلقة بالشرط الحدي  $T(1) = 1$  في العلاقة القودية (19.4) دون أن نضبط الشروط الحديثة المتعلقة بالبرهان بالاستقراء.

5-3.4

بيّن أن  $\Theta(n \lg n)$  هو الحل للعلاقة الحديثة "الدقيقة" (3.4) للفرز بالدمج.

6-3.4

بيّن أن حل العلاقة  $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$  هو  $O(n \lg n)$ .

7-3.4

يمكنك باستخدام الطريقة الرئيسة في المقطع 5.4 أن تبين أن حل العلاقة القودية  $T(n) = 4T(n/3) + n$  هو  $\Theta(n^{\log_3 4})$ .  $T(n) \leq cn^{\log_3 4}$  على برهان الفرضية  $T(n) \leq cn^{\log_3 4}$ . ثم بيّن كيف يسمح طرح حد أدنى مرتبة بإتمام البرهان بالتعويض.

8-3.4

يمكنك باستخدام الطريقة الرئيسة في المقطع 5.4 أن تبين أن حل العلاقة القودية  $T(n) = 4T(n/2) + n$  هو  $\Theta(n^2)$ .  $T(n) \leq cn^2$  على برهان الفرضية  $T(n) \leq cn^2$ . ثم بيّن كيف يسمح طرح حد أدنى مرتبة بإتمام البرهان بالتعويض.

9-3.4

حل العلاقة القودية  $T(n) = 3T(\sqrt{n}) + \log n$  بإجراء تغيير في التحويلات. يجب أن يكون حلك ملاصقاً بالمقارنة. ولا تعبأ بكون القيم طبيعية أو لا.

## 4.4 طريقة شجرة القُوْدِيَّة لحل العلاقات القُوْدِيَّة

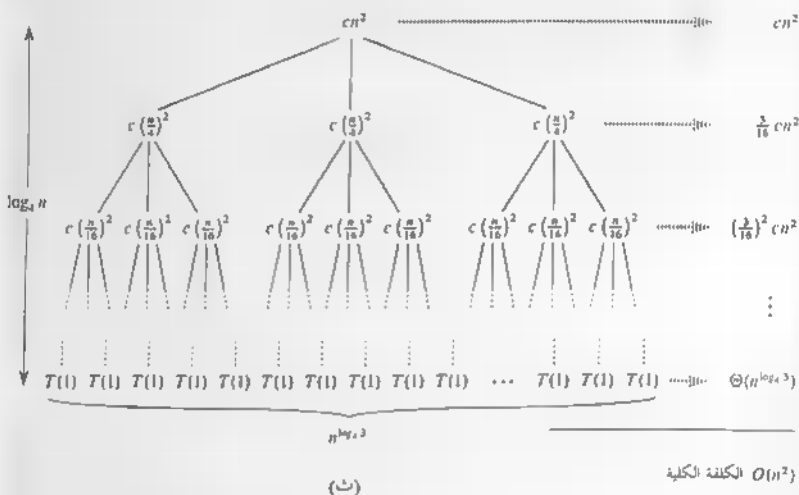
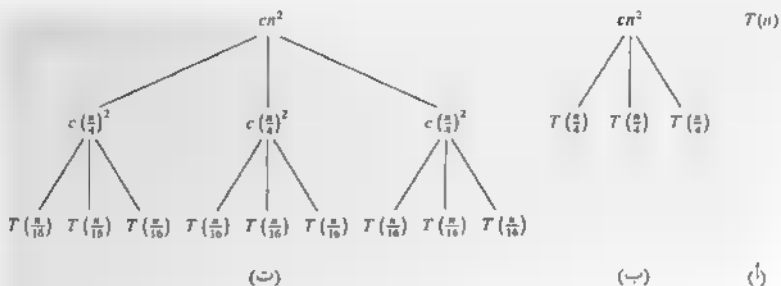
على الرغم من أن بمقدورك استخدام طريقة التعويض لتقدّم برهانًا موجزًا لصحة حل علاقة عُودِيَّة ما، إلا أنه قد يصعب عليك في بعض الأحيان التكهن بتخمين جيد، وغندها يعتبر رسم شجرة عُودِيَّة، كما فعلنا في تحليلنا للعلاقة القُوْدِيَّة للفرز بالدمج في المقطع 2.3.2، طريقة مباشرة للخروج بتخمين جيد. تُمثّل كل عقدة في شجرة القُوْدِيَّة *recursion tree*، كلفة مسألة جزئية واحدة موجودة في مكان ما داخل مجموعة الاستدعاءات القُوْدِيَّة. يجمع الكلف داخل كل مستوى من الشجرة لنحصل على مجموعة من التكاليف وفق المستويات، ثم نجمع هذه الكلف كلّها لنحسب الكلفة الكلّية لكل المستويات في الاستدعاءات القُوْدِيَّة.

أحسن ما تقوم به شجرة القُوْدِيَّة هي أنّا تولّد تخمينًا جيدًا، ليحري التحقق منه فيما بعد باستخدام طريقة التعويض. عندما نستخدم شجرة القُوْدِيَّة للحصول على تخمين جيد، يمكنك دائمًا التسامح مع قدر قليل من "عدم الدقة"، إذ إنك ستقوم لاحقًا بالتحقق من تخمينك. ولكن إذا كنت حريصًا جدًا وأنت ترسم شجرة القُوْدِيَّة، وتجمع التكاليف، يمكنك استخدامها برهانًا مباشرًا لحل العلاقة القُوْدِيَّة. سنستخدم شجرات القُوْدِيَّة، في هذا المقطع لتوليد تخمينات جيدة، وسنستخدمها في المقطع 6.4 لنبرهن مباشرة المبرهنة التي تكون الأساس للطريقة الرئيسة.

سنجد، على سبيل المثال، كيف يمكن لشجرة عُودِيَّة أن تعطي تخمينًا جيدًا للعلاقة القُوْدِيَّة  $T(n) = 3T(n/4) + \Theta(n^2)$ . نبدأ بالتركيز على البحث عن حدّ أعلى للحل. ولما كنا نعرف أنه لا أثر عادة لدوال الأسف والأرضيات في حل العلاقات القُوْدِيَّة (هذا مثال على عدم الدقة الذي يمكننا القبول به هنا)، فإننا سنسبب شجرة القُوْدِيَّة للعلاقة القُوْدِيَّة  $T(n) = 3T(n/4) + cn^2$  معلّنين بأن المعامل الثابت المستخدم هو  $c > 0$ .

يبين الشكل 5.4 اشتقاق شجرة القُوْدِيَّة للعلاقة  $T(n) = 3T(n/4) + cn^2$ . وللمسهولة، نفترض أن  $n$  من قوى 4 (مثال آخر عن عدم الدقة المقبول) بحيث تكون كل أحجام المسائل الجزئية أعدادًا صحيحة. يبيّن الجزء (أ) من الشكل:  $T(n)$ ، التي تُوسّع في الجزء (ب) إلى شجرة مكافئة تمثّل العلاقة القُوْدِيَّة. يمثّل الحد  $cn^2$  في الجذر الكلفة في أعلى مستوى من الاستدعاء القُوْدِي، وتمثّل الأشجار الفرعية الثلاث المتفرعة من الجذر التكاليف الناتجة عن المسائل الجزئية التي قياسها  $n/4$ . يبيّن الجزء (ت) هذه العملية مكررة خطوة أخرى بنشر كل عقدة كلفتها  $T(n/4)$  من الجزء (ب). وكلفة كل من الأبناء الثلاثة للجذر هي  $c(n/4)^2$ . وتتابع نشر كل عقدة في الشجرة بأن نرفعها إلى الأجزاء التي تتكون منها تبنا للعلاقة القُوْدِيَّة.

لما كانت أحجام المسائل الجزئية تتناقص بمعامل 4 كلما نزلنا من مستوى إلى المستوى الأدنى منه، كان لا بدّ لنا من الوصول إلى شرطٍ حدّي. ولكن كم يبعد الجذر عن مثل هذا الحد؟ إن حجم المسألة الجزئية لعقدة عمقها  $i$  هو  $n/4^i$ ، إذن سيصل حجم المسألة الجزئية إلى  $n = 1$  عندما يصبح  $n/4^i = 1$ ، أو ما يكافئه من أن  $i = \log_4 n$ . إذن يكون للشجرة  $\log_4 n + 1$  مستوى (على الأعماق  $0, 1, 2, \dots, \log_4 n$ ).



**الشكل 5.4** بناء شجرة عودية للعلاقة  $T(n) = 3T(n/4) + cn^2$ . يبين الجزء (أ) التي نشرها تدريجيًا في الأجزاء (ب)–(ث) لتكون شجرة العودية. يبلغ ارتفاع الشجرة المنشورة كليًا في الجزء (ث)  $\log_4 n$  (ونفيها  $\log_4 n + 1$  مستوى).

بعد ذلك نحدد كلفة كل مستوى من الشجرة. يضم كل مستوى ثلاثة أضعاف العقد الموجودة في المستوى الذي يعلوه، وبذلك يكون عدد العقد على العمق  $i$  مساويًا  $3^i$ . ولما كان حجم المسائل الجزئية ينقص وفق عامل يساوي 4 لكل مستوى نَنزل إليه بدءًا من الجذر، تكون كلفة كل عقدة على العمق  $i$ ، عندما  $i = 0, 1, 2, \dots, \log_4 n - 1$ ، مساوية  $c(n/4^i)^2$ . ويضرب هذه الكلفة بعدد العقد في المستوى  $i$ ، تكون الكلفة الكلية على العمق  $i$ ، عندما  $i = 0, 1, 2, \dots, \log_4 n - 1$ ، مساوية



لـ  $3^i c(n/4^i)^2 = (3/16)^i cn^2$  وبضم أدنى المستويات، وهو على العمق  $\log_4 n$ ، ما مقداره  $3^{\log_4 n} = n^{\log_4 3}$  عقدة، كل منها يساهم بكلفة  $T(1)$ ، إذن من أجل كلفة كلية مساوية لـ  $n^{\log_4 3} T(1)$ ، وهي  $\Theta(n^{\log_4 3})$ ، إذ أننا نفترض أن  $T(1)$  ثابت. نجمع الآن التكاليف على كل للمستويات لتحديد كلفة الشجرة بأكملها:

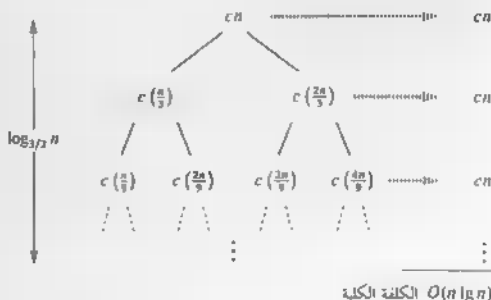
$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \dots + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\ &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{(3/16)^{\log_4 n} - 1}{(3/16) - 1} cn^2 + \Theta(n^{\log_4 3}) \quad ((5.أ)) \end{aligned}$$

تبدو الصيغة الأخيرة التي توصلنا إليها فوضوية بعض الشيء، وذلك حتى نتذكر مجددًا أنه بإمكاننا الاستفادة من القليل من عدم الدقة ونستخدم سلسلة هندسية متناقصة لا نهائية كحد أعلى. فإذا عدنا إلى الوراء خطوة واحدة وطبقنا للمعادلة (الملحق 6)، يكون لدينا:

$$\begin{aligned} T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{1}{1 - (3/16)} cn^2 + \Theta(n^{\log_4 3}) \\ &= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\ &= O(n^2). \end{aligned}$$

إذن، اشتققنا تخمينًا بأن  $T(n) = O(n^2)$  من علاقتنا القودية الأصلية  $T(n) = 3T(n/4) + \Theta(n^2)$ . تشكل معاملات الحد  $cn^2$  في هذا المثال سلسلة هندسية متناقصة، وباستخدام المعادلة (الملحق 6)، يمكن حد مجموع هذه المعاملات من الأعلى بالنايت 16/13. ولما كانت مساهمة الجذر في الكلفة الكلية هي  $cn^2$ ، فإن الجذر يساهم بجزء ثابت من هذه الكلفة. وبعبارة أخرى، تسيطر كلفة الجذر على الكلفة الكلية للشجرة.

إذًا،  $O(n^2)$  هو في الواقع حد أعلى للعلاقة القودية، وهذا ما سنتحقق منه بعد قليل، إذن يجب أن يكون هذا الحد ملائمًا. لماذا؟ يساهم الاستدعاء القودي الأول بكلفة  $\Theta(n^2)$ ، إذًا يجب أن يكون  $\Omega(n^2)$  حدًا أدنى للعلاقة القودية.



الشكل 6.4 شجرة عؤدية للعلاقة العؤدية  $T(n) = T(n/3) + T(2n/3) + cn$

بمقدورنا الآن أن نستخدم طريقة التعويض للتحقق من أن تخميننا صحيح، أي إن  $T(n) = O(n^2)$  هو حد أعلى للعلاقة العؤدية  $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$ . نريد أن نبين أن  $T(n) \leq dn^2$  لثابت  $d > 0$ . باستخدام الثابت  $c > 0$  نفسه الذي استخدمناه من قبل، يكون لدينا

$$\begin{aligned} T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2, \end{aligned}$$

حيث تكون الخطوة الأخيرة محققة مادام  $d \geq (16/3)c$ .

يبيّن الشكل 6.4 مثالاً آخر أشد تعقيداً لشجرة العؤدية للعلاقة

$$T(n) = T(n/3) + T(2n/3) + O(n).$$

(نحذف، ثانية، دالّي السقف والأرضية بهدف التبسيط). وكما سبق، نجعل  $c$  يمثل العامل الثابت في الحد  $O(n)$ . وعندما نجمع القيم لمستويات شجرة العؤدية للبيئة في الشكل، نحصل على القيمة  $cn$  لكل مستوى من المستويات. وأطول مسار بسيط من الجذر إلى ورقة هو  $n \leftarrow (2/3)n \leftarrow (2/3)^2 n \leftarrow \dots \leftarrow 1$ . ولما كان  $(2/3)^k n = 1$  عندما  $k = \log_{3/2} n$ ، فإن ارتفاع الشجرة يساوي  $\log_{3/2} n$ .

نتوقع بالحدس أن يكون حل العلاقة العؤدية هو على الأكثر عدد المستويات مضروباً بكلفة كل مستوى، أو  $O(cn \log_{3/2} n) = O(n \log n)$ . يبين الشكل 6.4 للمستويات العليا فقط من شجرة العؤدية، ولا تساهم كل المستويات بكلفة  $cn$ . لنأخذ كلفة الأوراق. لو كانت شجرة العؤدية هذه شجرة ثنائية كاملة ارتفاعها كل المستويات بكلفة  $cn$ . لكان هناك  $2^{\log_{3/2} n} = n^{\log_{3/2} 2}$  ورقة. ولما كانت كلفة كل ورقة ثابتة، ويجب أن تكون الكلفة

الكلفة للأوراق  $\Theta(n^{\log_{3/2} 2})$ ، ولما كان  $\log_{3/2} n$  ثابتاً أكبر تحاشاً من 1، فإن هذه الكلفة هي  $\omega(n \lg n)$ . ولكن شجرة الفؤدية هذه ليست شجرة ثنائية كاملة، ولذلك فإن عدد أوراقها أقل من  $n^{\log_{3/2} 2}$  ورقة. أضف إلى ذلك أن مزيداً من العقد الداخلية يخفي مع ابتعادنا عن الجذر. إذن، تساهم المستويات الأقرب إلى الأسفل في الكلفة الكلية بكلفة أقل من  $cn$ . فللمستويات في الأسفل تساهم بأقل من ذلك. يمكننا البحث عن حساب دقيق لكل الكلف، ولكن نذكر أننا نحاول فقط أن نأتي بتخمين نستخدمه فيما بعد في طريقة التعويض، فدعنا نتسامح مع قلة الدقة هذه، ونحاول أن نبين أن تخميناً من الشكل  $O(n \lg n)$  للحد الأعلى هو تخمين صحيح.

يمكننا في الواقع، استخدام طريقة التعويض للتحقق من أن  $O(n \lg n)$  هو حد أعلى لحل العلاقة الفؤدية. نبين فيما يلي أن  $T(n) \leq d n \lg n$  حيث  $d$  ثابت موجب مناسب. لدينا

$$\begin{aligned} T(n) &\leq T(n/3) + T(2n/3) + cn \\ &\leq d(n/3) \lg(n/3) + d(2n/3) \lg(2n/3) + cn \\ &= (d(n/3) \lg n - d(n/3) \lg 3) \\ &\quad + (d(2n/3) \lg n - d(2n/3) \lg(3/2)) + cn \\ &= d n \lg n - d((n/3) \lg 3 + (2n/3) \lg(3/2)) + cn \\ &= d n \lg n - d((n/3) \lg 3 + (2n/3) \lg 3 - (2n/3) \lg 2) + cn \\ &= d n \lg n - d n \lg 3 + (2n/3) \lg 2 + cn \\ &\leq d n \lg n . \end{aligned}$$

مادام  $d \geq c/(lg 3 - (2/3))$ ، إذن، لم تكن مضطرين لإجراء حساب أكثر دقة للكلف في شجرة الفؤدية.

## تعاريف

### 1-4.4

استخدم شجرة الفؤدية لتحديد حد أعلى مقارب جيد للعلاقة الفؤدية  $T(n) = 3T(\lfloor n/2 \rfloor) + n$ . استخدم طريقة التعويض للتحقق من جوابك.

### 2-4.4

استخدم شجرة الفؤدية لتحديد حد أعلى مقارب جيد للعلاقة الفؤدية  $T(n) = T(n/2) + n^2$ . استخدم طريقة التعويض للتحقق من جوابك.

### 3-4.4

استخدم شجرة الفؤدية لتحديد حد أعلى مقارب جيد للعلاقة الفؤدية  $T(n) = 4T(n/2 + 2) + n$ . استخدم طريقة التعويض للتحقق من جوابك.

## 4-4.4

استخدم شجرة القودية لتحديد حد أعلى مقارب جيد للعلاقة القودية  $T(n) = 2T(n-1) + 1$ . استخدم طريقة التعويض للتحقق من جوابك.

## 5-4.4

استخدم شجرة القودية لتحديد حد أعلى مقارب جيد للعلاقة القودية  $T(n) = T(n-1) + T(n/2) + n$ . استخدم طريقة التعويض للتحقق من جوابك.

## 6-4.4

برهن باستخدام شجرة القودية أن حل العلاقة القودية  $T(n) = T(n/3) + T(2n/3) + cn$  حيث  $c$  ثابت، هو  $\Omega(n \lg n)$ .

## 7-4.4

ارسم شجرة القودية للعلاقة  $T(n) = 4T(\lfloor n/2 \rfloor) + cn$ ، حيث  $c$  ثابت وأعط حلاً مقارباً ملائماً لحلها. تحقق - باستخدام طريقة التعويض - من الحد الذي تقترحه.

## 8-4.4

استخدم شجرة القودية لإعطاء حل ملائم بالمقارنة للعلاقة القودية  $T(n) = T(n-a) + T(a) + cn$ ، حيث  $a \geq 1$  و  $c > 0$  ثوابت.

## 9-4.4

استخدم شجرة القودية لإعطاء حل ملائم بالمقارنة للعلاقة القودية  $T(n) = T(an) + T((1-\alpha)n) + cn$ ، حيث  $\alpha$  ثابت محصور في المجال  $0 < \alpha < 1$  و  $c > 0$  ثابت أيضاً.

## 5.4 الطريقة الرئيسية لحل العلاقات القودية 5.4

نقدّم الطريقة الرئيسية master method "وصفة" لحل العلاقات القودية من الشكل

$$T(n) = aT(n/b) + f(n) \quad (20.4)$$

حيث  $a \geq 1$  و  $b > 1$  ثابتان و  $f(n)$  دالة موجبة بالمقارنة. لاستخدام الطريقة الرئيسية، نحتاج إلى تدكّر ثلاث حالات، يمكنك بعدها حل العديد من العلاقات القودية بسهولة مطلقة، غالباً بدون ورقة وقلم.

تصف العلاقة القودية في (20.4) زمن تنفيذ خوارزمية تقسم مسألة حجمها  $n$  إلى  $a$  مسألة جزئية، كل منها حجمه  $n/b$ ، حيث  $a$  و  $b$  ثابتان موجبان. نُحلّ المسائل الجزئية التي عددها  $a$  عُزودياً، كل منها في زمن  $T(n/b)$ . وتصف الدالة  $f(n)$  كلفة تقسيم المسألة وكلفة تجميع نتائج المسائل الجزئية. مثلاً تأخذ العلاقة

الغودية الناتجة عن خوارزمية شتراسن القيم  $a = 7$  و  $b = 2$  و  $f(n) = \Theta(n^2)$ .  
 إن العلاقة الغودية غير معروفة جيدًا من وجهة نظر الصحة التقنية، إذ قد لا يكون  $n/b$  عددًا طبيعيًا، إلا أن الاستعاضة عن كل من الحدود  $T(n/b)$  والتي عددها  $a$ ، سواء بـ  $T(\lfloor n/b \rfloor)$  أو بـ  $T(\lceil n/b \rceil)$  لا يؤثر على السلوك المقارب للعلاقة الغودية. (ستبرهن هذه الفرضية في المقطع القادم.) ولذلك عادةً ما نجد حذف دالتي الأرضية والسقف مناسبًا عند كتابة العلاقات الغودية "فرق-تسد" من هذا الشكل.

### المبرهنة الرئيسة

تعتمد الطريقة الرئيسة على المبرهنة التالية:

#### مبرهنة 1.4 (المبرهنة الرئيسة)

ليكن  $a \geq 1$  و  $b > 1$  ثابتين، وليكن  $f(n)$  دالة، وليكن  $T(n)$  معرّنة على الأعداد الطبيعية بالعلاقة الغودية

$$T(n) = aT(n/b) + f(n),$$

حيث نفتر  $n/b$  على أنها  $\lfloor n/b \rfloor$  أو  $\lceil n/b \rceil$ . وعندها يكون لـ  $T(n)$  الحدود التالية بالمقاربة:

1. إذا كان  $f(n) = O(n^{\log_b a - \epsilon})$  حيث  $\epsilon > 0$  ثابت ما، فإن  $T(n) = \Theta(n^{\log_b a})$ .
2. إذا كان  $f(n) = \Theta(n^{\log_b a} \lg n)$ ، فإن  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. إذا كان  $f(n) = \Omega(n^{\log_b a + \epsilon})$  حيث  $\epsilon > 0$  ثابت ما، وإذا كان  $af(n/b) \leq cf(n)$  حيث  $c < 1$  ثابت ما، و  $\blacksquare$  كبيرة كفاية، فإن  $T(n) = \Theta(f(n))$ .

دعونا، قبل أن نطبق المبرهنة الرئيسة على بعض الأمثلة، نغني بعض الوقت في محاولة فهم هذه المبرهنة. نحن نقارن في كلٍّ من الحالات الثلاث، الدالة  $f(n)$  بالدالة  $n^{\log_b a}$ . يمتدّد حل العلاقة الغودية حديسيًا تبعًا لأكثر الدالتين. إذا كانت الدالة  $n^{\log_b a}$  هي الأكبر، كما في الحالة 1، فإن الحل هو  $T(n) = \Theta(n^{\log_b a})$ . وإذا كانت الدالة  $f(n)$  هي الأكبر، كما في الحالة 3، فإن الحل هو  $T(n) = \Theta(f(n))$ . وإذا كانت الدالتان من القياس نفسه، كما في الحالة 2، فإننا نضرب بعامل لوغارتمي، ويكون الحل  $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(f(n) \lg n)$ .

يقف بيننا وبين هذا الجس، أن نكون واعين لبعض التفاصيل التقنية. ففي الحالة الأولى، لا يكفي أن تكون  $f(n)$  أصغر من  $n^{\log_b a}$ ، بل يجب أن تكون أصغر منها حدوديًا *polynomially*. أي أن تكون  $f(n)$  أصغر بالمقاربة من  $n^{\log_b a}$  بعامل  $n^\epsilon$  حيث  $\epsilon > 0$  ثابت. ولا يكفي في الحالة الثالثة أن تكون الدالة  $f(n)$  أكبر من  $n^{\log_b a}$ ، بل يجب أن تكون أكبر منها حدوديًا، ويجب أن تحقق شرط "الانتظام" أي  $af(n/b) \leq cf(n)$ . وعمومًا تحقق هذا الشرط معظم الدوال المحدودة حدوديًا التي سندرسها.

لاحظ أن الحالات الثلاث لا تشمل كل حالات  $f(n)$  للمكينة! فهناك فجوة تفصل بين الحالتين 1 و 2، عندما تكون  $f(n)$  أصغر من  $n^{\log_2 a}$ ، ولكن ليس أصغر حدوديًا. وبالمثل هناك فجوة تفصل بين الحالتين 2 و 3، عندما تكون  $f(n)$  أكبر من  $n^{\log_2 a}$  ولكن ليست أكبر حدوديًا. إذا وقعت الدالة  $f(n)$  في إحدى هاتين الفجوتين الناصبتين، أو إذا لم يتحقق شرط الانتظام في الحالة 3، لا يمكنك استخدام الطريقة الرئيسية لحل العلاقة التّؤدية.

### استخدام الطريقة الرئيسية

لاستخدام الطريقة الرئيسية، نحدد من المبرهنة الرئيسية أية حالة (إن وجدت) تنطبق ونكتب الجواب. لنأخذ كمثال أول

$$T(n) = 9T(n/3) + n.$$

لدينا، في حالة هذه العلاقة التّؤدية،  $a = 9$  و  $b = 3$  و  $f(n) = n$ ، وبهذا لدينا  $n^{\log_2 a} = n^{\log_2 9} = \Theta(n^2)$ . ولما كان  $f(n) = O(n^{\log_2 9 - \epsilon})$ ، حيث  $\epsilon = 1$ ، يمكننا تطبيق الحالة 1 من المبرهنة الرئيسية، ونستنتج أن الحل هو  $T(n) = \Theta(n^2)$ . لنأخذ الآن

$$T(n) = T(2n/3) + 1,$$

حيث  $a = 1$  و  $b = 3/2$  و  $f(n) = 1$  و  $n^{\log_2 a} = n^{\log_2 1} = n^0 = 1$ . تنطبق الحالة 2، لأن  $T(n) = \Theta(\lg n)$  و  $f(n) = \Theta(n^{\log_2 a}) = \Theta(1)$ . وهكذا يكون حل العلاقة التّؤدية  $T(n) = \Theta(\lg n)$ . لدينا في حالة العلاقة التّؤدية

$$T(n) = 3T(n/4) + n \lg n,$$

$a = 3$  و  $b = 4$  و  $f(n) = n \lg n$  و  $n^{\log_2 a} = n^{\log_2 3} = O(n^{0.793})$ ، ولما كان  $\Omega(n^{\log_2 3 + \epsilon})$ ، حيث  $\epsilon \approx 0.2$ ، فإن الحالة 3 تنطبق إذا استطعنا أن نبين أن  $f(n)$  تحقق شرط الانتظام. فإذا كانت  $n$  كبيرة كفاية، يكون لدينا  $cf(n) = 3(n/4) \lg(n/4) \leq (3/4)n \lg n = cf(n)$ ، حيث  $c = 3/4$ . وهكذا يكون حل العلاقة التّؤدية اعتماديًا على الحالة 3:  $T(n) = \Theta(n \lg n)$ . إن الطريقة الرئيسية لا تنطبق على العلاقة التّؤدية

$$T(n) = 2T(n/2) + n \lg n,$$

رغم أن لها الصيغة المناسبة:  $a = 2$  و  $b = 2$  و  $f(n) = n \lg n$  و  $n^{\log_2 a} = n$ . قد تعتقد خطأ أن الحالة 3 يجب أن تنطبق، لأن  $f(n) = n \lg n$  أكبر من  $n^{\log_2 a} = n$  بالمقارنة، ولكن المشكلة أنها ليست أكبر حدوديًا. فالنسبة  $f(n)/n^{\log_2 a} = (n \lg n)/n = \lg n$  أصغر من  $n^\epsilon$  بالمقارنة مهما كانت قيمة الثابت الموجب  $\epsilon$ . ومن ثم تقع العلاقة التّؤدية في الفجوة الفاصلة بين الحالة 2 والحالة 3. (انظر التمرين 2-6.4)

لإيجاد حل لها.

لنستخدم الطريقة الرئيسية لحل العلاقات التّؤدية التي رأيناها في المقتطين 1.4 و 2.4. إن العلاقة التّؤدية (7.4)

$$T(n) = 2T(n/2) + \Theta(n) ,$$

توصّف أزمنة تنفيذ فرق-تسد لكلّ من مسألة الصغيفة الجزئية العظمى والفقر بالدمج. (كما هو الحال دائماً، أممنا ذكر الحالة القاعدية في العلاقة التّؤدية.) لدينا هنا  $a = 2$  و  $b = 2$  و  $f(n) = \Theta(n)$ ، وبذلك يكون لدينا  $n^{\log_b a} = n^{\log_2 2} = n$ . تنطبق الحالة 2، لأن  $f(n) = \Theta(n)$ ، وبذلك يكون لدينا الحل  $T(n) = \Theta(n \lg n)$ .

أما العلاقة التّؤدية (17.4)

$$T(n) = 8T(n/2) + \Theta(n^2) ,$$

فتوصّف زمن تنفيذ خوارزمية فرق-تسد الأولى التي رأيناها لجداء المصفوفات. لدينا الآن  $a = 8$  و  $b = 2$ ، وبذلك يكون لدينا  $n^{\log_b a} = n^{\log_2 8} = n^3$ . ولما كانت  $n^3$  أكبر من  $f(n)$  حدوديًا (أي  $f(n) = O(n^{3-\epsilon})$  حيث  $\epsilon = 1$ )، فإن الحالة 1 تنطبق، ويكون  $T(n) = \Theta(n^3)$ . وأخيراً، لنأخذ العلاقة التّؤدية (18.4)

$$T(n) = 7T(n/2) + \Theta(n^2) ,$$

التي تصف زمن تنفيذ خوارزمية شتراسن. لدينا هنا  $a = 7$  و  $b = 2$ ، وبذلك يكون لدينا  $n^{\log_b a} = n^{\log_2 7}$ . بإعادة كتابة  $\log_2 7$  على أنّها  $\lg 7$ ، ونذكر أن  $2.81 < \lg 7 < 2.80$  نرى أن  $f(n) = O(n^{1.87-\epsilon})$  حيث  $\epsilon = 0.8$ ، وتنطبق الحالة 1 ثانية، ويكون لدينا الحل  $T(n) = \Theta(n^{1.87})$ .

تعارين

1-5.4

استخدم الطريقة الرئيسية لتعطي حدودًا ملاصقة بالمقارنة للعلاقات التّؤدية التالية:

أ.  $T(n) = 2T(n/4) + 1$ .

ب.  $T(n) = 2T(n/4) + \sqrt{n}$ .

ت.  $T(n) = 2T(n/4) + n$ .

ث.  $T(n) = 2T(n/4) + n^2$ .

2-5.4

يرغب الأستاذ قيصر Caesar بتطوير خوارزمية جداء مصفوفات أسرع من خوارزمية شتراسن بالمقارنة.

مستخدم خوارزميته طريقة فرق-تسد، مقسمة إلى أجزاء قياسها  $n/4 \times n/4$ ، وتستغرق خطوات التفريق والتجميع مقًا زمنًا  $\Theta(n^2)$ . يحتاج لتحديد عدد المسائل الجزئية التي يجب أن تنشأ خوارزميته حتى يتفوق على خوارزمية شتراسن. إذا نشأت خوارزميته  $a$  مسألة جزئية، فإن العلاقة القودية لزمن التنفيذ  $T(n)$  تصبح  $T(n) = aT(n/4) + \Theta(n^2)$ . ما هي أكبر قيمة صحيحة لـ  $a$  تكون بها خوارزمية الأستاذ فيسر أسرع من خوارزمية شتراسن بالمقارنة؟

### 3-5.4

استخدم الطريقة الرئيسة لتبين أن حل العلاقة القودية للبحث الثنائي  $T(n) = T(n/2) + \Theta(1)$  هو  $T(n) = \Theta(\lg n)$ . (انظر التمرين 5-3.2 لوصف البحث الثنائي.)

### 4-5.4

هل يمكن تطبيق الطريقة الرئيسة على العلاقة القودية  $T(n) = 4T(n/2) + n^2 \lg n$ ؟ علّل. أعطِ حدًا أعلى بالمقارنة لهذه العلاقة القودية.

### \* 5-5.4

لنأخذ شرط الانتظام  $af(n/b) \leq cf(n)$  حيث  $c < 1$  ثابت ما، الذي هو جزء من الحالة 3 من المبرهنة الرئيسة. أعطِ مثالاً على ثابتين  $a \geq 1$  و  $b > 1$  ودالة  $f(n)$  تحقق كل الشروط في الحالة 3 من المبرهنة الرئيسة ما عدا شرط الانتظام.

## \* 6.4 برهان المبرهنة الرئيسة

يتضمن هذا المقطع برهان المبرهنة الرئيسة (المبرهنة 1.4). ولا حاجة إلى أن تفهم البرهان لكي تطبق هذه المبرهنة.

إن البرهان يتألف من جزأين. يحلّل الجزء الأول العلاقة القودية "الرئيسة master" (20.4) مع الافتراض المبسط أن  $T(n)$  معرفة فقط عند القوى الصحيحة لـ  $b > 1$ ، أي عندما  $n = 1, b, b^2, \dots$ . ويعطي هذا الجزء كل الحس لازم للاقتناع بصحة المبرهنة الرئيسة. ويبيّن الجزء الثاني كيف يمكن تعميم التحليل السابق على كل الأعداد الطبيعية  $n$ ، وذلك بتطبيق تقنيات رياضية لمسألة التعامل مع الأرصيات والأسقف.

سنسرف في هذا المقطع، بعض الشيء في استخدام التدوين للمقارب، وذلك باستخدامه أحياناً لوصف سلوك دوال معرفة على قوى  $b$  الصحيحة فقط. نذكّر أن تعاريف التدوينات للمقاربة تتطلب برهان الحدود لكل الأعداد الكبيرة كفاية، وليس لقوى  $b$  فقط. ولكن، لما كان بمقدورنا وضع تدوينات مقاربة جديدة تنطبق على المجموعة  $\{b^i : i = 0, 1, 2, \dots\}$ ، عوضاً عن الأعداد الطبيعية، فإن هذا التجاوز ضئيل الأهمية.

ومع ذلك، يجب أن نكون دائماً متنبهين عندما نستخدم التدوين للمقارب على نطاق محدود حتى لا



تتوصل إلى استنتاجات خاطئة. فعلى سبيل المثال، إذا برهنا أن  $T(n) = O(n)$  عندما تكون  $n$  قوة صحيحة لـ 2، فإن هذا لا يضمن أن يكون  $T(n) = O(n)$ . إذ يمكن تعريف الدالة  $T(n)$  كما يلي

$$T(n) = \begin{cases} n & \text{if } n = 1, 2, 4, 8, \dots, \\ n^2 & \text{otherwise} \end{cases}$$

حيث إن أفضل حد أعلى يمكن برهانه هنا هو  $T(n) = O(n^2)$ . لذلك، وبسبب هذا النوع من العواقب السيئة، فإننا لن نستخدم التدوين للمقارب أبدًا على نطاق محدود دون أن نشير إلى ذلك بوضوح تام في السياقات.

#### 1.6.4 البرهان في حالة القوى الصحيحة

يحلّل الجزء الأول من برهان المبرهنة الرئيسة العلاقة التّؤدّية (20.4)

$$T(n) = aT(n/b) + f(n) ,$$

الواردة في الطريقة الرئيسة، مع الفرض بأن  $\blacksquare$  هي قوة صحيحة لـ  $b > 1$ ، حيث ليس بالضرورة أن تكون  $b$  عددًا طبيعيًا. ينقسم التحليل إلى ثلاث توططات. تحتل الأولى مسألة حل العلاقة التّؤدّية العامة إلى مسألة حساب عبارة فيها مجموع. وتحدّد التوططة الثانية حدودًا على هذا المجموع. وتجمع التوططة الثالثة سابقتها معًا لتهرّن نسخة من المبرهنة الرئيسة عندما تكون  $n$  إحدى القوى الصحيحة لـ  $b$ .

##### توططة 2.4

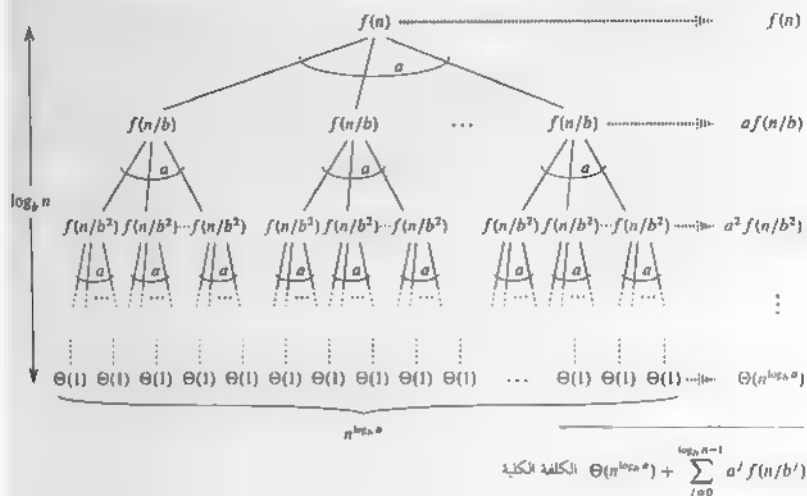
ليكن  $\blacksquare \geq 1$  و  $b > 1$  ثابتين، ولكن  $f(n)$  دالة موجبة معرّفة على القوى الصحيحة لـ  $b$ . نعرّف  $T(n)$  على القوى الصحيحة لـ  $b$  بالعلاقة التّؤدّية

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ aT(n/b) + f(n) & \text{if } n = b^i , \end{cases}$$

حيث  $a$  عدد صحيح موجب. إذن

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) . \quad (21.4)$$

**البرهان** نستخدم شجرة التّؤدّية المذكورة في الشكل 7.4. إن كلفة جذر الشجرة هي  $f(n)$ ، وله  $a$  أبناء، كلفة كلٍّ منهم  $f(n/b)$ . (من المناسب أن نتعامل مع  $a$  على أنه عدد طبيعي، وخاصة عندما نعاين شجرة التّؤدّية، إلا أن ذلك غير ضروري رياضيًا.) ولكل من هؤلاء الأبناء  $\blacksquare$  أبناء، وهكذا هناك  $a^2$  عقدة على البعد 2 من الجذر، كلفة كل منها  $f(n/b^2)$ . وفي الحالة العامة، هناك  $a^j$  عقدة على البعد  $j$  من الجذر، وكل منها كلفتها  $f(n/b^j)$ . كلفة كل ورقة هي  $\Theta(1) = T(1)$ ، وكل ورقة هي على عمق  $\log_b n$ ، وذلك لأن  $n/b^{\log_b n} = 1$  وهناك  $a^{\log_b n} = n^{\log_b a}$  ورقة في الشجرة.



**الشكل 7.4** شجرة القُوْدَة التي تولّدُها العلاقة  $T(n) = aT(n/b) + f(n)$ . إن هذه الشجرة هي شجرة كاملة ذات  $a$  فرعاً من كل عقدة مع  $n^{\log_b a}$  ورقة، ارتفاعها  $\log_b n$ . نجد إلى اليمين من كل مستوى كلفته، وتغطي المعادلة (21.4) مجموع هذه التكاليف.

يمكن الوصول إلى المعادلة (21.4) بمجموع تكاليف كل للمستويات في الشجرة، كما هو مبين في الشكل. كلفة العقد الداخلية على العمق  $j$  هو  $a^j f(n/b^j)$  وهكذا يكون المجموع لكل المستويات الداخلية

$$\sum_{j=0}^{\log_b n - 1} a^j f(n/b^j).$$

يمثل هذا المجموع، في خوارزمية فرق-تسد التي أنتجت، تكاليف تقسيم المسائل إلى مسائل جزئية ومن ثمّ مجموع هذه المسائل الجزئية. وتكون كلفة كل الأوراق  $\Theta(n^{\log_b a})$ ، وهي كلفة حلّ مسألة جزئية حجم كل منها 1.

تقابل الحالات الثلاث في المبرهنة الرئيسة، بالنظر إلى شجرة القُوْدَة الحالات التي تكون فيها الكلفة الكلية للشجرة (1) جلّها من كلف الأوراق (2) موزّعة بالتساوي في جميع مستويات الشجرة أو (3) جلّها في كلفة الجذر.

يصف المجموع في العلاقة (21.4) كلفة خطوات التقسيم والتجميع في خوارزمية فرق-تسد التي أنتجت العلاقة. وتزوّدنا للمبرهنة التالية بحدود مقاربة لنمو هذا المجموع.

## 3.4 توطئة

ليكن  $a \geq 1$  و  $b > 1$  ثابتين، ولتكن  $f(n)$  دالة موجبة معروفة على القوى الصحيحة لـ  $b$ . يمكن إعطاء حدّ مقارب على القوى الصحيحة لـ  $b$  لدالة  $g(n)$  معروفة على القوى الصحيحة لـ  $b$  من الشكل

$$g(n) = \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j) \quad (22.4)$$

وذلك تبعاً للحالات التالية:

1. إذا كان  $f(n) = O(n^{\log_b a - \epsilon})$ ، حيث  $\epsilon > 0$  ثابت ما، فإن  $g(n) = O(n^{\log_b a})$ .
2. إذا كان  $f(n) = \Theta(n^{\log_b a} \lg n)$ ، فإن  $g(n) = \Theta(n^{\log_b a} \lg n)$ .
3. إذا كان  $af(n/b) \leq cf(n)$ ، حيث  $c < 1$  ثابت ما، فإن  $g(n) = \Theta(f(n))$ ، لجميع قيم  $n$  الكبيرة كفاية.

**البرهان** لدينا في الحالة 1:  $f(n) = O(n^{\log_b a - \epsilon})$ ، وهذا يقتضي أن  $f(n/b^j) = O((n/b^j)^{\log_b a - \epsilon})$ . بالتعويض في المعادلة (22.4) نجد

$$g(n) = O\left(\sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon}\right). \quad (23.4)$$

نجد هذا المجموع في التوسيم- $O$  بتفريق الحدود وبالتبسيط، فينتج عن ذلك سلسلة هندسية متزايدة:

$$\begin{aligned} \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a - \epsilon} &= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} \left(\frac{ab^\epsilon}{b^{\log_b a}}\right)^j \\ &= n^{\log_b a - \epsilon} \sum_{j=0}^{\log_b n - 1} (b^\epsilon)^j \\ &= n^{\log_b a - \epsilon} \left(\frac{b^{\epsilon \log_b n} - 1}{b^\epsilon - 1}\right) \\ &= n^{\log_b a - \epsilon} \left(\frac{n^\epsilon - 1}{b^\epsilon - 1}\right). \end{aligned}$$

ولما كان  $b$  و  $\epsilon$  ثابتين، فاستطاعتنا إعادة كتابة العبارة الأخيرة كالآتي  $n^{\log_b a - \epsilon} O(n^\epsilon) = O(n^{\log_b a})$ .  
وبتعويض هذه العبارة بالمجموع في المعادلة (23.4) نجد

$$g(n) = O(n^{\log_b a}),$$

وبهذا نكون قد برهننا الحالة 1.

ولما كانت الحالة 2 نفترض أن  $f(n) = \Theta(n^{\log_b a})$  فإن  $f(n/b^j) = \Theta((n/b^j)^{\log_b a})$  وبالتعويض في المعادلة (24.4) نجد

$$g(n) = \Theta\left(\sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}\right). \quad (24.4)$$

نحذف هذا المجموع بالتدوين  $\Theta$  كما في الحالة 1، ولكن لا نحصل هذه المرة على سلسلة هندسية، بل نكتشف هنا أن كل حدود المجموع متماثلة:

$$\begin{aligned} \sum_{j=0}^{\log_b n-1} a^j \left(\frac{n}{b^j}\right)^{\log_b a} &= n^{\log_b a} \sum_{j=0}^{\log_b n-1} \left(\frac{a}{b^{\log_b a}}\right)^j \\ &= n^{\log_b a} \sum_{j=0}^{\log_b n-1} 1 \\ &= n^{\log_b a} \log_b n. \end{aligned}$$

وبتعويض هذه العبارة بالمجموع في المعادلة (24.4) نجد

$$\begin{aligned} g(n) &= \Theta(n^{\log_b a} \log_b n) \\ &= \Theta(n^{\log_b a} \lg n), \end{aligned}$$

وبهذا نكون قد برهننا الحالة 2.

نُقرن الحالة 3 بطريقة مماثلة. فلما كان  $f(n)$  يظهر في التعريف (22.4) للدالة  $g(n)$ ، ولما كانت كل حدود الدالة  $g(n)$  موجبة، يمكننا استنتاج أن  $g(n) = \Omega(f(n))$  لقوى  $b$  الصحيحة. نفترض في نص التوطئة أن  $af(n/b) \leq cf(n)$  حيث  $c < 1$  ثابت ما، وقيم  $n$  كلها كبيرة كفاية. نعيد كتابة هذه الفرضية كما يلي  $f(n/b) \leq (c/a)f(n)$ ، ونكرر ذلك  $z$  مرة، فنجد  $f(n/b^j) \leq (c/a)^j f(n)$ ، أو ما يكافئه  $a^j f(n/b^j) \leq c^j f(n)$ ، حيث نفترض أن قيم  $n$  في عمليات التكرار كبيرة كفاية. تحقق جميع الحدود ما عدا عدد ثابت منها هذه المتراجحة، ويكون أصغر الحدود التي لا يحققها  $n/b^{j-1}$ ، والذي يتحقق من أجله  $a^j f(n/b^{j-1}) = O(1)$ .

وبالتعويض في المعادلة (22.4) وبالتبسيط نحصل على سلسلة هندسية، ولكن حدود هذه السلسلة متناقصة، خلافاً للحالة 1. نستخدم الحد  $O(1)$  لنختزل فيه كل الحدود التي لا تحقق فرضيتنا بأن  $n$  كبيرة كفاية:

$$g(n) = \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

$$\begin{aligned}
&\leq \sum_{j=0}^{\log_b n-1} c^j f(n) + O(1) \\
&\leq f(n) \sum_{j=0}^{\infty} c^j + O(1) \\
&= f(n) \left( \frac{1}{1-c} \right) + O(1) \\
&= O(f(n)) ,
\end{aligned}$$

وذلك لأن  $c$  ثابت. وهكذا يمكننا أن نستنتج أن  $g(n) = \Theta(f(n))$  لقوى  $b$  الصحيحة. وبإتمام برهان الحالة 3، يكون قد تم برهان التوطئة. ■

يمكننا الآن برهان نسخة من المعرنة الرئيسة في الحالة التي تكون فيها  $n$  قوة صحيحة لـ  $b$ .

#### توطئة 4.4

ليكن  $a \geq 1$  و  $b > 1$  ثابتين، ولتكن  $f(n)$  دالة موجبة معرفة على القوى الصحيحة لـ  $b$ . نعرف  $T(n)$  على القوى الصحيحة لـ  $b$  بالعلاقة القودية التالية:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 , \\ aT(n/b) + f(n) & \text{if } n = b^i , \end{cases}$$

حيث  $i$  عدد طبيعي. وعندما يمكن حد  $T(n)$  بالمقارنة عند القوى الصحيحة لـ  $b$  كالآتي:

1. إذا كان،  $f(n) = O(n^{\log_b a - \epsilon})$  حيث  $\epsilon > 0$  ثابت ما، فإن  $T(n) = \Theta(n^{\log_b a})$ .
2. إذا كان  $f(n) = \Theta(n^{\log_b a} \lg n)$ ، فإن  $T(n) = \Theta(n^{\log_b a} \lg n)$ .
3. إذا كان  $f(n) = \Omega(n^{\log_b a + \epsilon})$  حيث  $\epsilon > 0$  ثابت ما، وإذا كان  $af(n/b) \leq cf(n)$  حيث  $c < 1$  ثابت ما، و  $n$  كبيرة كفاية، فإن  $T(n) = \Theta(f(n))$ .

**البرهان** نستخدم الحدود في التوطئة 3.4 لتقدير المجموع (21.4) من التوطئة 2.4. في الحالة 1 نجد

$$\begin{aligned}
T(n) &= \Theta(n^{\log_b a}) + O(n^{\log_b a}) \\
&= \Theta(n^{\log_b a}) ,
\end{aligned}$$

وفي الحالة 2،

$$\begin{aligned}
T(n) &= \Theta(n^{\log_b a}) + \Theta(n^{\log_b a} \lg n) \\
&= \Theta(n^{\log_b a} \lg n) .
\end{aligned}$$

وفي الحالة 3،

$$\begin{aligned} T(n) &= \Theta(n^{\log_b a}) + \Theta(f(n)) \\ &= \Theta(f(n)) , \end{aligned}$$

■

$$f(n) = \Omega(n^{\log_b a + \epsilon})$$

## 2.6.4 الأرضيات والأسقف

لإنهاء برهان المبرنة الرئيسة، علينا أن نوسّع تحليلنا ليشمل الحالة التي تكون فيها الأرضيات والأسقف floors and ceilings مستخدمة في العلاقة القودية الرئيسة، بحيث تكون العلاقة القودية معرفة على كل الأعداد الصحيحة، وليس على القوى الصحيحة لـ  $b$  فقط. من السهل الحصول على حد أدنى للدالة

$$T(n) = aT(\lceil n/b \rceil) + f(n) \quad (25.4)$$

وعلى حد أعلى للدالة

$$T(n) = aT(\lfloor n/b \rfloor) + f(n) \quad (26.4)$$

إذ يمكن دفع المتراجحة  $n/b \leq \lceil n/b \rceil$  إلى الحالة الأولى للحصول على النتيجة المطلوبة، ويمكن دفع المتراجحة  $n/b \geq \lfloor n/b \rfloor$  إلى الحالة الثانية. إن إيجاد حد أدنى للعلاقة القودية (26.4) يتطلب استخدام الطريقة نفسها لإيجاد حد أعلى للعلاقة القودية (25.4)، ولذلك، فإننا سنكتفي بتقديم حد للأخيرة.

نغير شجرة القودية في الشكل 7.4 لتولّد شجرة القودية في الشكل 8.4. ومع نزولنا في شجرة القودية، نحصل على سلسلة من الاستدعاءات القودية على المخلّطات.

$$\begin{aligned} n , \\ \lceil n/b \rceil , \\ \lceil \lceil n/b \rceil / b \rceil , \\ \lceil \lceil \lceil n/b \rceil / b \rceil / b \rceil , \\ \vdots \end{aligned}$$

نسّم  $n_j$  العنصر ذا الرقم  $j$  في المتتالية، حيث

$$n_j = \begin{cases} n & \text{if } j = 0 , \\ \lceil n_{j-1}/b \rceil & \text{if } j > 0 . \end{cases} \quad (27.4)$$

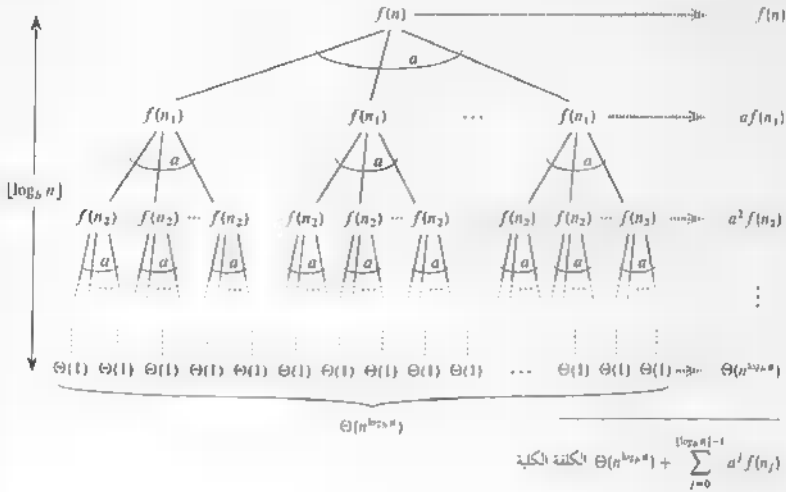
غرضنا الأول هو أن نحدّد العمق  $k$  بحيث يكون  $n_k$  ثابتًا، باستخدام المتراجحة  $x \leq x + 1$ ، نحصل على

$$n_0 \leq n ,$$

$$n_1 \leq \frac{n}{b} + 1 ,$$

$$n_2 \leq \frac{n}{b^2} + \frac{1}{b} + 1 ,$$

$$n_3 \leq \frac{n}{b^3} + \frac{1}{b^2} + \frac{1}{b} + 1 ,$$



**الشكل 8.4** شجرة التكرارية التي تولدها  $T(n) = aT(n/b) + f(n)$ . نُضَدُّ التكراري  $n_j$  معطى لي  
المعادلة (27.4).

وعموماً، لدينا

$$\begin{aligned} n_j &\leq \frac{n}{b^j} + \sum_{i=0}^{j-1} \frac{1}{b^i} \\ &< \frac{n}{b^j} + \sum_{i=0}^{\infty} \frac{1}{b^i} \\ &= \frac{n}{b^j} + \frac{b}{b-1} \end{aligned}$$

وإذا جعلنا  $j = \log_b n$  نحصل على

$$\begin{aligned} n_{\lceil \log_b n \rceil} &< \frac{n}{b^{\lceil \log_b n \rceil}} + \frac{b}{b-1} \\ &< \frac{n}{b^{\log_b n - 1}} + \frac{b}{b-1} \\ &= \frac{n}{n/b} + \frac{b}{b-1} \\ &= b + \frac{b}{b-1} \\ &= O(1) \end{aligned}$$

وهكذا نرى أنه عند العمق  $\lfloor \log_b n \rfloor$ ، يكون حجم المسألة على الأكثر ثابتاً.

ومن الشكل 8.4 نرى أنَّ

$$T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\lfloor \log_b n \rfloor - 1} a^j f(n_j) \quad (28.4)$$

وهي علاقة مشابهة كثيراً للمعادلة (21.4)، باستثناء كون  $n$  عدداً صحيحاً اختيارياً، ولا ينحصر في مجموعة القوى الصحيحة لـ  $b$ .

يمكننا الآن حساب قيمة المجموع

$$g(n) = \sum_{j=0}^{\lfloor \log_b n \rfloor - 1} a^j f(n_j) \quad (29.4)$$

من المعادلة (28.4)، وبطريقة مشابهة لبرهان التوطئة 3.4. نبدأ بالحالة 3: إذا كان  $af((n/b)) \leq cf(n)$  عندما  $n > b + b/(b-1)$ ، حيث  $c < 1$  ثابت، فهذا يستتبع أن يكون  $a^j f(n_j) \leq c^j f(n)$  ولهذا السبب يمكن حساب المجموع في المعادلة (29.4) غنائاً كما في التوطئة 3.4. وفي الحالة 2، لدينا  $f(n) = \Theta(n^{\log_b a})$ . إذا استطعنا أن نبين أن  $f(n) = O((n/b^j)^{\log_b a}) = O(n^{\log_b a} / a^j)$ ،  $f(n_j) = O(n^{\log_b a} / a^j)$ ، كان بمقدورنا إتمام البرهان في الحالة 2 من التوطئة 3.4. لاحظ أن  $\lfloor \log_b n \rfloor \leq j$  يقتضي  $b^j / n \leq 1$ . إن الحد  $f(n) = O(n^{\log_b a})$  يقتضي وجود ثابت  $c > 0$  بحيث يكون، مهما كان  $n_j$  كبيراً كفاية،

$$\begin{aligned} f(n_j) &\leq c \left( \frac{n}{b^j} + \frac{b}{b-1} \right)^{\log_b a} \\ &= c \left( \frac{n}{b^j} \left( 1 + \frac{b^j}{n} \cdot \frac{b}{b-1} \right) \right)^{\log_b a} \\ &= c \left( \frac{n^{\log_b a}}{a^j} \right) \left( 1 + \left( \frac{b^j}{n} \cdot \frac{b}{b-1} \right) \right)^{\log_b a} \\ &\leq c \left( \frac{n^{\log_b a}}{a^j} \right) \left( 1 + \frac{b}{b-1} \right)^{\log_b a} \\ &= O \left( \frac{n^{\log_b a}}{a^j} \right). \end{aligned}$$

لأن  $c(1 + b/(b-1))^{\log_b a}$  ثابت. وهكذا نكون قد برهننا الحالة 2. وأما برهان الحالة 1، فمماثل له تقريباً. والفكرة الأساسية هي أن نبرهن الحد  $f(n_j) = O((n/b^j)^{\log_b a - \epsilon})$ ، وهذا مماثل للبرهان المقابل في الحالة 2، إلا أن العمليات الجبرية أكثر دقة.

لقد برهننا الآن الحدود العليا في المبرهنة الرئيسة على كل الأعداد الصحيحة  $n$ ، أما برهان الحدود الدنيا

فمشابه لما سبق.



## تمارين

\* 1-6.4

أعطِ عبارة بسيطة ودقيقة لـ  $n_j$  في المعادلة (27.4) في الحالة التي يكون فيها  $b$  عددًا صحيحًا موجبًا بدلاً من أن يكون عددًا حقيقيًا اختياريًا.

= 2-6.4

بيِّن أنه إذا كان  $f(n) = \Theta(n^{\log_a b} \lg^k n)$ ، حيث  $k \geq 0$ ، فإن حل العلاقة التّؤدية العامة هو  $T(n) = \Theta(n^{\log_a b} \lg^{k+1} n)$ . وللتبسيط، افترض في تحليلك على القوى الصحيحة لـ  $b$ .

\* 3-6.4

بيِّن أن في الحالة 3 من البرهنة الرئيسة نكررًا في الشروط، إذ إن شرط الانتظام  $af(n/b) \leq cf(n)$  حيث  $c < 1$  ثابت ما، يقتضي وجود ثابت  $\epsilon > 0$  بحيث يكون  $f(n) = \Omega(n^{\log_a b + \epsilon})$ .

## مسائل

## 1-4 أمثلة على العلاقات التّؤدية

أعطِ حدودًا عليا ودنيا لـ  $T(n)$  لكل من العلاقات التّؤدية التالية. افترض أن  $T(n)$  ثابت عندما  $n \leq 2$ . اجعل حدودك ملاحظة قدر الإمكان، وعلّل أجوبتك.

$$أ. \quad T(n) = 2T(n/2) + n^4$$

$$ب. \quad T(n) = T(7n/10) + n$$

$$ت. \quad T(n) = 16T(n/4) + n^2$$

$$ث. \quad T(n) = 7T(n/3) + n^2$$

$$ج. \quad T(n) = 7T(n/2) + n^2$$

$$ح. \quad T(n) = 2T(n/4) + \sqrt{n}$$

$$خ. \quad T(n) = T(n-2) + n^2$$

## 2-4 تكاليف تمرير المتوسطات

نفترض في هذا الكتاب أن تمرير المتوسطات في استدعاءات الإجرائيات يستغرق زمنًا ثابتًا، ولو كنا نمرّر صفيقة من  $N$  عنصرًا. إن هذه الفرضية صحيحة في معظم الأنظمة لأن ما يُمرّر هو مؤشر على هذه الصفيقة، وليس الصفيقة نفسها. تدرس هذه المسألة ملائمة استخدام ثلاث استراتيجيات لتمرير المتوسطات.

- . صيغة ممرّة باستخدام مؤشر، الزمن هو:  $\text{Time} = \Theta(1)$ .
- . صيغة ممرّة بالنسخ، الزمن هو:  $\text{Time} = \Theta(N)$ ، حيث  $N$  حجم الصيغة.
- . صيغة ممرّة بنسخ الجزء الذي قد تنفذ إليه الإجراءية المستدعاة فقط، الزمن هو:  $\text{Time} = \Theta(q - p + 1)$  إذا كانت الصيغة الجزئية الممررة هي  $A[p \dots q]$ .

أ. لتأخذ خوارزمية البحث الثنائي القودية للعثور على عدد في صيغة مرتبة (انظر التمرين 5-3.2). أعط العلاقات القودية لأزمنة تنفيذ البحث الثنائي في أسوأ الحالات عندما تمرر الصيغيات باستخدام كل طريقة من الطرق السابقة، وأعط حدودًا عليا جيّدة لحلول العلاقات القودية. افترض أن  $N$  حجم المسألة الأصلية، و  $\equiv$  حجم المسألة الجزئية.

ب. أعد الجزء (أ) على خوارزمية MERGE-SORT من المقطع 1.3.2.

### 3- أمثلة إضافية على العلاقات القودية

عط حدودًا عليا ودنيا مقاربة لـ  $T(n)$  لكل من العلاقات القودية التالية. افترض أن  $T(n)$  ثابت عندما تكون  $n$  صغيرة كفاية. اجعل حدودك ملاصقة قدر الإمكان، وعمل أجوتك

- أ.  $T(n) = 4T(n/3) + n \lg n$ .
- ب.  $T(n) = 3T(n/3) + n/\lg n$ .
- ت.  $T(n) = 4T(n/2) + n^2\sqrt{n}$ .
- ث.  $T(n) = 3T(n/3 - 2) + n/2$ .
- ج.  $T(n) = 2T(n/2) + n/\lg n$ .
- ح.  $T(n) = T(n/2) + T(n/4) + T(n/8) + n$ .
- خ.  $T(n) = T(n - 1) + 1/n$ .
- د.  $T(n) = T(n - 1) + \lg n$ .
- ذ.  $T(n) = T(n - 2) + 1/\lg n$ .
- ر.  $T(n) = \sqrt{n}T(\sqrt{n}) + n$ .

### 4-4 أعداد فيوناتشي

تقدّم هذه المسألة خصائص جديدة لأعداد فيوناتشي المعرّفة بالعلاقة القودية (22.3). سوف نستخدم طريقة

الدوال المولدة لحل علاقة فيوناتشي التوافقية. نعرف  $F$  الدالة المولدة *generating function* (أو سلسلة القوى الصورية *formal power series*) كالتالي:

$$F(z) = \sum_{i=0}^{\infty} F_i z^i$$

$$F(z) = 0 + z + z^2 + 2z^3 + 3z^4 + 5z^5 + 8z^6 + 13z^7 + 21z^8 + \dots$$

حيث  $F_i$  هو عدد فيوناتشي ذو الترتيب  $i$ .

$$أ. \text{ بَيِّنْ أَنَّ } F(z) = z + zF(z) + z^2F(z)$$

ب. بَيِّنْ أَنَّ

$$\begin{aligned} F(z) &= \frac{z}{1-z-z^2} \\ &= \frac{z}{(1-\phi z)(1-\hat{\phi} z)} \\ &= \frac{1}{\sqrt{5}} \left( \frac{1}{1-\phi z} - \frac{1}{1-\hat{\phi} z} \right). \end{aligned}$$

جبت

$$\phi = \frac{1+\sqrt{5}}{2} = 1.61803 \dots$$

و

$$\hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.61803 \dots$$

ت. برهن أن

$$F(z) = \sum_{i=0}^{\infty} \frac{1}{\sqrt{5}} (\phi^i - \hat{\phi}^i) z^i.$$

ث. استخدم الجزء (ت) لتبرهن أن  $F_i = \phi^i / \sqrt{5}$  عندما  $i > 0$ ، مدوّرة إلى أقرب عدد صحيح. (تلميح: لاحظ أن  $|\hat{\phi}| < 1$ .)

#### 3-4 اختيار رقائقات VLSI

لدى الأستاذ ديوجينيس  $n$  رقاقة VLSI<sup>1</sup> من المفترض أن تكون متماثلة: والقادرة من حيث المبدأ على أن تُستخدم لاختبار بعضها بعضاً. تُستخدم لوحة اختبار الأستاذ رقائقتين في الوقت نفسه. عندما تكون اللوحة

<sup>1</sup> تعني VLSI "very large scale integration"، وهي تقانة رقائقات الدارات للتكاملة المستخدمة في تصنيع معظم المعالجات الصغرى اليوم.

محتملة، نختبر كل رقاقة الرقاقة الأخرى وتبين حالتها (جيدة أو سيئة). فإذا كانت الرقاقة جيدة، فإنها تعطي بياناً صحيحاً دائماً عن حالة الرقاقة الأخرى، غير أن الأستاذ لا يمكن أن يثق بمجواب شرعية سيئة. لذا، فإن نتائج الاختبار الأربعة للممكنة هي كالتالي:

بيان الرقاقة A	بيان الرقاقة B	النتيجة
■ في حالة جيدة	A في حالة جيدة	كلتاها في حالة جيدة، أو كلتاها في حالة سيئة
B في حالة جيدة	A في حالة سيئة	واحدة على الأقل في حالة سيئة
B في حالة سيئة	A في حالة جيدة	واحدة على الأقل في حالة سيئة
B في حالة سيئة	A في حالة سيئة	واحدة على الأقل في حالة سيئة

أ. بين أنه إذا كانت  $n/2$  رقاقة على الأقل في حالة سيئة، فإن الأستاذ لا يستطيع تحديد الرقاقات الجيدة مهما كانت الاستراتيجية التي يستخدمها باستخدام هذا النوع من الاختبارات الثنائية. افترض أن الرقاقات السيئة تتواطأ لتخدع الأستاذ.

ب. لتأمل في مسألة العثور على رقاقة جيدة واحدة من بين  $n$  رقاقة، بافتراض وجود أكثر من  $n/2$  رقاقة جيدة. بين أن  $[n/2]$  اختباراً ثنائياً كافٍ لاختزال المسألة إلى مسألة أخرى بنصف الحجم تقريباً.

ت. بين أنه يمكن معرفة الرقاقات الجيدة بـ  $\Theta(n)$  اختباراً ثنائياً، بافتراض وجود أكثر من  $n/2$  رقاقة جيدة. أعط العلاقة القودية التي تصف عدد الاختبارات ثم حلها.

#### 6-4 صفيفات مونج

نقول عن صفيف  $A$  مؤلفة من  $n \times n$  عدداً حقيقياً إنها صفيفة مونج *Monge Array* إذا تحقّق

$$A[i, j] + A[k, l] \leq A[i, l] + A[k, j] .$$

أيما كان  $i$  و  $j$  و  $k$  و  $l$  حيث  $1 \leq i < k \leq n$  و  $1 \leq j < l \leq n$ .

وبعبارة أخرى، عندما نختار صفين وعمودين من صفيفة مونج، وننظر إلى العناصر الأربعة على تقاطع السطرين والعمودين، يكون مجموع العنصرين الأعلى الأيسر والأدنى الأيمن أصغر أو يساوي مجموع العنصرين الأدنى الأيسر والأعلى الأيمن. مثلاً، الصفيفة التالية هي صفيفة مونج:

10	17	13	28	23
17	22	16	29	23
24	28	22	34	24
11	13	6	17	7
45	44	32	37	23
36	33	19	21	6
75	66	51	53	34

أ. برهن أن صيغة ما هي صيغة مونج إذا وفقط إذا كان لدينا

$$A[i, j] + A[i + 1, j + 1] \leq A[i, j + 1] + A[i + 1, j] ,$$

أيًا كان  $i = 1, 2, \dots, m - 1$  و  $j = 1, 2, \dots, n - 1$ .

(تلميح: فيما يتعلق بالجزء "إذا"، استخدم الاستقراء على الأسطر والأعمدة على نحو منفصل.)

ب. الصيغة التالية ليست صيغة مونج. عرِّ عنصرًا واحدًا لجعلها صيغة مونج. (تلميح: استخدم الجزء أ.)

37	23	22	32
21	6	7	10
53	34	30	31
32	13	9	6
43	21	15	8

ت. ليكن  $f(i)$  مؤشر العمود الذي يحوي أصغر عنصر إلى أقصى اليسار في السطر  $i$ . برهن أن

$$f(1) \leq f(2) \leq \dots \leq f(m)$$

ث. لدينا هنا وصف لخوارزمية فرز-تسد تُحسب أصغر عنصر إلى أقصى اليسار في كل سطر من صيغة

$$\text{مونج } A \text{ بُعدها } m \times n$$

ابن مصفوفة جزئية  $A'$  من  $A$ ، تتألف من الأسطر الزوجية من  $A$ . حدِّدْ عُدُّيًا أصغر عنصر إلى أقصى اليسار في كل سطر من  $A'$ . ثم احسب أصغر عنصر إلى أقصى اليسار في الأسطر المفردة من  $A$ .

اشرح كيف يمكن حساب أصغر عنصر إلى أقصى اليسار في الأسطر المفردة من  $A$  في زمن  $O(m + n)$ . (بافتراض أننا نعرف أصغر عنصر إلى اليسار في الأسطر الزوجية.)

ج. اكتب العلاقة العودية التي تصف زمن تنفيذ الخوارزمية الموصَّفة في الجزء (ث). بيِّن أن حلها هو

$$O(m + n \log m)$$

## ملاحظات الفصل

تعود بدايات طريقة فرز-تسد لتصميم الخوارزميات إلى ما قبل 1962 في مقال لـ Ofman و Karatsuba [194]. إلا أنها - نبيلاً لـ Heideman و Johnson و Burras [163] - قد تكون استخدمت قبل ذلك بكثير، فقد صمم C. F. Gauss أول خوارزمية تحويل فورييه السريع في 1805، حيث تجزئ صياغة Gauss المسألة إلى مسائل جزئية أصغر لتركب حلولها معاً.

إن مسألة الصيغة الجزئية الصغرى المذكورة في المقطع 1.4 ما هي إلا تعديلٌ طفيف على مسألة درسها

Bentley [43] في الفصل السابع.

لقد أحدثت خوارزمية شتراسن Strassen [325] الكثير من الإثارة عندما نُشرت في عام 1969؛ إذ إن قلة قليلة فقط هم الذين كانوا يتخيلون إمكانية إيجاد خوارزمية أسرع بالمقارنة من إجراء SQUARE-MATRIX-MULTIPLY الأساسي. ولقد أُحرزت تحسينات على الحد الأعلى لجداء المصفوفات منذ ذلك الحين. إن أكثر الخوارزميات فعالية بالمقارنة لحساب جداء مصفوفتين  $n \times n$  حتى يومنا هذا هي الخوارزمية المنسوبة إلى Coppersmith و Winograd [79]، وهي تحقق زمن تنفيذ  $O(n^{2.376})$ . أما أفضل حد أدنى معروف، فهو الحد البديهي  $\Omega(n^2)$  (هو بديهي لأنه يجب ملء  $n^2$  عنصراً في مصفوفة الجداء).

أما من وجهة النظر العملية، فإن خوارزمية شتراسن ليست على الأغلب الطريقة المفضلة لحساب جداء المصفوفات، للأسباب الأربعة التالية:

1. العامل الثابت المخفي  $\Theta(n^{1.87})$  لزمن تنفيذ خوارزمية شتراسن أكبر من العامل الثابت الموجود في  $\Theta(n^3)$  لزمن تنفيذ إجراء SQUARE-MATRIX-MULTIPLY.
2. عندما تكون المصفوفات متخلخلة sparse، فإن الطرق المصممة لهذا النوع من المصفوفات أسرع.
3. لا تتمتع خوارزمية شتراسن بالاستقرار الرقمي نفسه لخوارزمية SQUARE-MATRIX-MULTIPLY. وبعبارة أخرى، إن الدقة المحدودة للعمليات الحسابية المحوسبة على القيم غير الصحيحة تسبب في تراكم أخطاء أكبر في خوارزمية شتراسن منها في خوارزمية SQUARE-MATRIX-MULTIPLY.
4. تستهلك المصفوفات الجزئية للتكؤنة في مستويات الاستدعاءات القؤدية الذاكرة.

في عام 1990 تقريباً، خُفِّت الأبحاث من عمق أثر السببين الأخيرين. فقد برهن Higham [167] أن الفرق في الاستقرار الرقمي كان مبالغاً فيه؛ فعلى الرغم من أن خوارزمية شتراسن غير مستقرة رقمياً كفاية في بعض التطبيقات، إلا أنها في الحدود المقبولة في تطبيقات أخرى. يناقش Bailey و Lee و Simon [32] تقنيات للحد من متطلبات الذاكرة في خوارزمية شتراسن.

عملياً، نستخدم التحيزات السريعة لجداء المصفوفات الكثيفة خوارزمية شتراسن للمصفوفات التي تتجاوز أحجامها "نقطة تجاوز"، وتتحول إلى طريقة أبسط عندما يقل حجم للمسألة الجزئية عن نقطة التجاوز هذه. إن القيمة الدقيقة لنقطة التجاوز تتعلق كثيراً بالنظام الحاسوبي. وقد أعطت الدراسات التي بُجِّهت تأثير الذاكرة السريعة cache والنقل عبر أنابيب pipelining نقاط تجاوز منخفضة حتى  $n = 8$  (تبعاً لـ Higham [167]) أو  $n = 12$  (تبعاً لـ Huss-Lederman وآخرون [186]). فيما طوّر Nicolau و D'Alberto [82] منهجاً تكييفياً لتحديد نقطة التجاوز، وذلك بتطبيق تجارب معيارية benchmarking على الخزمة البرمجية عند تنصيبها. ووجدوا نقاط تجاوز على أنظمة متنوعة تقع ما بين  $n = 400$  و  $n = 2150$ ، ولم يتمكنوا من إيجاد نقطة تجاوز على نظام أو نظامين.

بدأت دراسة العلاقات العُودِيَّة بأكثر في العام 1202 على يد فيوناتشي L. Fibonacci، الذي سُميت أعداد فيوناتشي باسمه. وأدخل دومافر A. De Moivre طريقة الدوال المولَّدة لحلِّ العلاقات العُودِيَّة (انظر المسألة 4-4). الطريقة الرئيسة مستفاد من Bentley و Haken و Saxe [44]، التي تقدِّم طريقة موسَّعة مبررة بالتمرين 2-6.4. يبيِّن كل من Knuth [209] و Liu [237] كيف يمكن حل العلاقات العُودِيَّة الخطيَّة باستخدام طريقة الدوال المولَّدة. ويضَمُّ كل من Purdom و Brown [287] و Graham و Knuth و Patashnik [152] نقاشًا موسَّعًا حول حل العلاقات العُودِيَّة.

قدَّم العديد من الباحثين، ومنهم Akra و Bazzi [13]، و Roura [299] و Verma [346] و Yap [360] طرقًا لحلِّ علاقات عُودِيَّة من غط فرق-تسد أعمَّ من تلك التي يمكن حلِّها باستخدام الطريقة الرئيسة. ونشرح هنا نتيجة أعمال Akra و Bazzi التي عدَّلها Leighton [228]، والتي تنطبق على علاقات عُودِيَّة من الشكل

$$T(x) = \begin{cases} \Theta(1) & \text{if } 1 \leq x \leq x_0, \\ \sum_{i=1}^k a_i T(b_i x) + f(x) & \text{if } x > x_0. \end{cases} \quad (30.4)$$

حيث:

- $x \geq 1$  عدد حقيقي؛
- $x_0$  عدد ثابت بحيث يكون  $x_0 \geq 1/b_i$  و  $x_0 \geq 1/(1 - b_i)$  حيث  $i = 1, 2, \dots, k$ ؛
- $a_i$  ثابت موجب حيث  $i = 1, 2, \dots, k$ ؛
- $b_i$  ثابت في المجال  $0 < b_i < 1$  حيث  $i = 1, 2, \dots, k$ ؛
- $k \geq 1$  عدد صحيح ثابت،
- $f(x)$  دالة موجبة تحقق شرط النمو الحدودي *Polynomial-growth condition*: يوجد ثابتان موجبان  $c_1$  و  $c_2$  حيث، مهما كان  $x \geq 1$ ، و  $i = 1, 2, \dots, k$ ، ومهما كان  $u$  حيث  $b_i x \leq u \leq x$ ، يكون لدينا  $c_1 f(x) \leq f(u) \leq c_2 f(x)$ . (إذا كان  $|f'(x)|$  محدودًا من الأعلى بكثير حدود ما لا  $x$ ، فإن الدالة  $f(x)$  تحقق شرط النمو الحدودي. على سبيل المثال نحقق الدالة  $f(x) = x^\alpha \lg^\beta x$  هذا الشرط مهما كان الثابتان الحقيقيان  $\alpha$  و  $\beta$ .)

وعلى الرغم من أنه لا يمكن تطبيق الطريقة الرئيسة على علاقات عُودِيَّة مثل  $T(n) = T(\lfloor n/3 \rfloor) + T(\lfloor 2n/3 \rfloor) + O(n)$ ، فإنه يمكن تطبيق طريقة Akra-Bazzi. ولحل العلاقة العُودِيَّة (30.4)، فإننا نوجد أولاً العدد الحقيقي الوحيد  $p$  بحيث يكون  $\sum_{i=1}^k a_i b_i^p = 1$ . (إن  $p$  موجود دائمًا) وعندما يكون حل العلاقة العُودِيَّة هو

$$T(n) = \Theta\left(x^p \left(1 + \int_1^x \frac{f(u)}{u^{p+1}} du\right)\right).$$

قد يكون من الصعوبة بمكان استخدام طريقة Akra-Bazzi، ولكنها تفيد في حل علاقات عودية تنمذج تقسيم المسألة إلى مسائل جزئية مختلفة الحجم جوهريًا. أما الطريقة الرئيسة، فأسهل في الاستخدام، إلا أنه لا يمكن تطبيقها إلا عندما تكون أحجام المسائل الجزئية متساوية.



## 5 التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة

يعرف هذا الفصل بالتحليل الاحتمالي والخوارزميات ذات العشوائية المضافة probabilistic analysis and randomized algorithms. فإذا لم تكن على معرفة بأسس نظرية الاحتمالات، فينبغي أن تقرأ الملحق-ت، الذي يستعرض هذه المادة. وسنعود في هذا الكتاب عدّة مرات إلى التحليل الاحتمالي والخوارزميات ذات العشوائية للمضافة.

### 1.5 مسألة التوظيف

افترض أنك تريد أن توظف مساعدًا جديدًا في المكتب، وقد أخفقت محاولاتك السابقة للتوظيف، فقررت أن تلجأ إلى وكالة توظيف. سترسل لك وكالة التوظيف مرشحًا كل يوم، فتقابل ذلك الشخص ثم تقرّر توظيفه أو عدم توظيفه. وعليك أن تسدد لوكالة التوظيف عمولة صغيرة مقابل إجراء مقابلة مع متقدّم ما. لكن سيكون توظيف متقدّم ما أكثر كلفة إذ عليك أن تستغني عن مساعدك الحالي وأن تسدد عمولة كبيرة إلى وكالة التوظيف. وتضمن لك الوكالة أن تقدم لك على الدوام الشخص الأفضل للمهمة المتاحة لديها. ولهذا السبب قررت أنه - بعد مقابلة كل متقدّم - إذا كان هذا المتقدّم أفضل من مساعدك الحالي، فإنك ستستغني عن المساعد الحالي وتوظف المتقدّم الجديد. لا مانع لديك من تسديد ثمن هذه الاستراتيجية، إلا أنك ترغب في تقدير هذا الثمن.

يُعرّ الإجراء HIRE-ASSISTANT، المعطى هنا، عن استراتيجية التوظيف هذه بشبه رماز. ونفترض أن المرشحين لوظيفة مساعد المكتب مرقّمون من 1 إلى  $n$ . يفترض الإجراء أنك قادر، بعد مقابلة المرشح  $i$  على تحديد كون هذا المرشح  $i$  هو أفضل مرشح قابلته حتى ذلك الوقت. ينشئ الإجراء، بهدف النهاية، مرشحًا فعليًا، رقمه 0 هو الأسوأ من بين كل المرشحين الآخرين.

HIRE-ASSISTANT( $n$ )

- 1  $best = 0$  // candidate 0 is a least-qualified dummy candidate
- 2 for  $i = 1$  to  $n$
- 3 interview candidate  $i$

- 4 if candidate  $i$  is better than candidate  $best$   
 5  $best = i$   
 6 hire candidate  $i$

يختلف نموذج الكلفة لهذه المسألة عن النموذج للمشروع في الفصل 2. نحن لا نتم هنا بزم تنفيذ HIRE-ASSISTANT، بل بالكلفة الناتجة عن المقابلات والتوظيف. قد يبدو ظاهرياً أن تحليل كلفة هذه الخوارزمية مختلف تماماً عن تحليل زمن تنفيذ الفرز بالدمج مثلاً، إلا أن طرق التحليل المستخدمة هي نفسها سواء أكنّا نحلل كلفة أم زمن تنفيذ. ففي كلتا الحالتين، نحن نعدّ عدد مرات تنفيذ عمليات أساسية معينة. للمقابلة كلفة منخفضة، ولكن  $c_i$ ، على حين أن للتوظيف كلفة مرتفعة، ولكن  $c_h$ . وليكن  $m$  عدد الأشخاص الموفّقين. عندها تكون الكلفة الكلية للمقابلة لهذه الخوارزميات هي  $O(c_i n + c_h m)$ . ومهما كان عدد الأشخاص الذين نوظفهم، فإننا نقابل دائماً  $n$  مرشحاً، وهكذا فإن كلفة المقابلات  $c_i n$  ستترتب دائماً علينا. ولذلك فإننا سنركز على تحليل  $c_h m$ ، كلفة التوظيف. فهذه القيمة تتغير مع كل تنفيذ للخوارزمية. يفيد هذا السيناريو بوصفه نموذجاً لمنهجية عمل محسوب سائلة. فكثيراً ما نحتاج إلى إيجاد القيمة العظمى أو الصغرى لمتالية ما بدراسة كل عنصر فيها والاحتفاظ "بالفائز" الحالي. إن مسألة التوظيف تتمزج مدى تكرار تغيير رؤيتنا للعنصر الفائز حالاً.

### تحليل أسوأ الحالات

نقوم في أسوأ الحالات عملياً بتوظيف كل مرشح نقابله. ويتحقق ذلك عندما يرد المرشحون بالترتيب التصاعدي من حيث الكفاءة، وفي هذه الحالة نوظف  $n$  مرة، بكلفة توظيف كلية تبلغ  $O(c_h n)$ . طبقاً لا يرد المرشحون دائماً بالترتيب التصاعدي من حيث الكفاءة، وفي الحقيقة، لا علم لنا بترتيب ورودهم، وليس لنا أية قدرة على التأثير على هذا الترتيب، ولهذا السبب، من الطبيعي أن تتساوى عمّا نتوقع حدوثه في الحالة الاعتيادية أو الوسطى.

### التحليل الاحتمالي

**التحليل الاحتمالي probabilistic analysis** هو استخدام الاحتمالات في تحليل المسائل، والأكثر شيوعاً هو أن نستخدم التحليل الاحتمالي لتحليل زمن تنفيذ خوارزمية ما، ولكننا نستعمله أحياناً لتحليل مقادير أخرى، مثل كلفة التوظيف في إجراء HIRE-ASSISTANT. وللقيام بالتحليل الاحتمالي علينا أن نستخدم معرفتنا بتوزع المُدخلات الاحتمالي، أو أن نفترض فرضيات بشأنه. ثم نحلّل خوارزمتنا، لحساب زمن التنفيذ المتوقع، ونحسب التوقع على توزيع المُدخلات للممكنة. وهكذا، فإننا نقوم عملياً بحساب متوسط زمن التنفيذ على كل المُدخلات الممكنة. عندما نتحدث عن زمن تنفيذ مماثل لهذا الزمن، فإننا نشير إليه على أنه زمن التنفيذ في الحالة الوسطى *average-case running time*.

يجب أن نكون حذرين عند تحديد توزيع المدخلات الاحتمالي. لأنه في بعض المسائل، يكون افتراض بعض الفرضيات بشأن مجموعة المُدخلات الممكنة منطقيًا، ويمكننا عندها استخدام التحليل الاحتمالي كطريقة لتصميم خوارزمية فعالة، وكوسيلة للتعق في فهم المسألة. أما في مسائل أخرى، فليس بمقدورنا توصيف توزيع معقول للدخل، ولا يمكننا في هذه الحالة استخدام التحليل الاحتمالي.

يمكننا، فيما يخص مسألة التوظيف، افتراض أن المتقدمين يأتون وفق ترتيب عشوائي. ولكن ماذا يعني ذلك في هذه المسألة؟ نفترض أنه بمقدورنا مقارنة أي مرشحين وتحديد أيٍّ منهما هو الأنسب، أي إن هناك ترتيبًا شاملًا للمرشحين. (انظر تعريف الترتيب الشامل في الملحق ب.) وهكذا يمكننا إعطاء كل مرشح مرتبة هي عدد وحيد بين 1 و  $n$ ، وذلك باستخدام  $rank(i)$  للإشارة إلى مرتبة المتقدم  $i$ ، وباعتماد فرضية أن المرتبة الأعلى تقابل متقدمًا أكثر كفاءة. إن القائمة المرتبة  $\{rank(1), rank(2), \dots, rank(n)\}$  هي تبديل على القائمة  $\{1, 2, \dots, n\}$ . إن قولنا بأن المتقدمين يأتون وفق ترتيب عشوائي يكافئ أن نقول إنه من الممكن أن تكون قائمة المراتب هي أي تبديل من تبديل الأعداد من 1 إلى  $n$ ، والتي يبلغ عددها  $n!$ ، وذلك باحتمال متساوٍ. أو بطريقة أخرى، نقول إن المراتب تشكل تبديلًا عشوائيًا منتظمًا *uniform random permutation*، أي إن كل التباديل للمكئة، وعددها  $n!$ ، ترد باحتمال متساوٍ. يحتوي المقطع 2.5 تحليلًا احتماليًا لمسألة التوظيف.

### الخوارزميات ذات العشوائية المضافة

نحتاج، لاستخدام التحليل الاحتمالي، إلى بعض المعلومات عن توزيع المُدخلات. وفي كثير من الأحيان، لا نعرف إلا القليل عن توزيعها. وحتى إن كان لدينا بعض العلم بهذا التوزيع، فقد لا نكون قادرين على نمذجة هذه المعرفة بنموذج محوسبة. ومع ذلك، يمكننا في كثير من الأحيان، استخدام الاحتمالات والعشوائية كأدوات لتصميم الخوارزميات وتحليلها، بأن نجعل سلوك جزء من الخوارزمية عشوائيًا.

قد يبدو، في مسألة التوظيف، أن المرشحين يُرسلون إلينا وفق ترتيب عشوائي. ولكن، ليس بمقدورنا أن نعرف إذا كانوا يُرسلون كذلك فعلاً. ولهذا السبب، يجب أن نزيد في التحكم في ترتيب مقابلات المرشحين، كي نتمكن من تطوير خوارزمية ذات عشوائية مضافة لمسألة التوظيف. لذا، سنغير النموذج قليلاً. سنقول أن لدى وكالة التوظيف  $n$  مرشحًا، وأنها ترسل لنا سلفًا قائمةً بالمرشحين. وفي كل يوم، نحن نختار عشوائيًا المرشح الذي سنقابله. ومع أننا لا نعرف شيئًا عن المرشحين (ما عدا أسماءهم)، فقد أجرينا تغييرًا كبيرًا. فبدلاً من أن نعتمد على التخمين بأن المرشحين سيأتون بترتيب عشوائي، زدنا تحكمًا في العملية، وفرضنا ترتيبًا عشوائيًا.

عمومًا، نقول عن خوارزمية إنها ذات عشوائية مضافة *randomized* إذا كان سلوكها يتحدد، إضافة إلى الدخل، بالاعتماد على قيم يولدها مولّد أعداد عشوائية *random-number generator*. سنفترض أن في حوزتنا مولّد أعداد عشوائية *RANDOM*، وأن الاستدعاء  $RANDOM(a, b)$  يعيد عددًا طبيعيًا بين  $a$  و  $b$ ،

يمكن أن يكون  $a$  أو  $b$ ، وبحيث تكون كل الأعداد متساوية الاحتمال. فمثلاً،  $\text{RANDOM}(0,1)$  يعطي 0 باحتمال  $1/2$  و 1 باحتمال  $1/2$ . ويعيد استدعاء  $\text{RANDOM}(3,7)$  إحدى القيم 3 أو 4 أو 5 أو 6 أو 7، كل منها باحتمال  $1/5$ . إن كل عدد طبيعي يعيده  $\text{RANDOM}$  مستقل عن الأعداد المولدة في استدعاءات سابقة. يمكنك أن تتخيل  $\text{RANDOM}$  وكأنك ترمي حجر ترد له  $(b - a + 1)$  وجهًا لتحصل على عرجه (تقدّم معظم بيانات البرمجة عمليًا مولد أعداد شبه عشوائية *pseudorandom-number generator*: وهو خوارزمية حتمية تعيد أعدادًا "تبدو" إحصائيًا وكأنها عشوائية).

عندما ندرس زمن تنفيذ خوارزمية ذات عشوائية مضافة، نأخذ توقع زمن التنفيذ تبعًا للتوزيع الاحتمالي للقيم التي يعيدها مولد الأعداد العشوائية. غيّر هذه الخوارزميات عن تلك التي يكون فيها الدخول عشوائيًا بأن نشير إلى زمن تنفيذ خوارزمية ذات عشوائية مضافة على أنه **زمن التنفيذ المتوقع** *expected running time*. وعمومًا، نتحدث عن زمن التنفيذ في الحالة الوسطى عندما يكون التوزيع الاحتمالي على مدخلات الخوارزمية، وتتحدث عن زمن التنفيذ المتوقع عندما تقوم الخوارزمية نفسها باختيار خيارات عشوائية.

## تمارين

### 1-1.5

بيّن أن الفرضية التي تقول إننا دائمًا قادرون على تحديد أي مرشح هو الأفضل في السطر 4 من الإجراء **HIRE-ASSISTANT** نقضي أن نعرف ترتيبًا شاملاً على مراتب المرشحين.

### \* 2-1.5

وصفّ تنجيلاً للإجراء  $\text{RANDOM}(a, b)$  تقوم فيه فقط باستدعاءات  $\text{RANDOM}(0, 1)$ . ما هو زمن التنفيذ المتوقع لإجرائيتك باعتبارها دالة لـ  $a$  و  $b$ ؟

### ■ 3-1.5

افترض أنك تريد عرجًا يساوي 0 باحتمال  $1/2$  و 1 باحتمال  $1/2$ . وفي حوزتك إجراء **BIASED-RANDOM**، عرجه إما 1 أو 0. نخرج 1 باحتمال  $p$ ، و 0 باحتمال  $1 - p$ ، حيث  $0 < p < 1$ ، ولكنك لا تعرف قيمة  $p$ . أعط خوارزمية تستخدم **BIASED-RANDOM** باعتباره إجراءً فرعيًا، وتعيد جوابًا غير منحاز، أي تعيد 0 باحتمال  $1/2$  و 1 باحتمال  $1/2$ . ما هو زمن التنفيذ المتوقع لخوارزمتك بوصفها دالة لـ  $p$ ؟

## 2.5 المتحولات العشوائية المؤشرة

سنستخدم، لتحليل العديد من الخوارزميات، ومنها مسألة التوظيف، متحولات المؤشرات العشوائية التي تمثّل طريقة مناسبة للتحويل بين الاحتمالات والتوقعات. لنفترض أن لدينا فضاء عينة  $S$  وحدثًا  $A$ ، فيكون **المؤشر**

المشوائي<sup>1</sup> indicator random variable  $I\{A\}$  الموافق للحدث  $A$  معرفًا كما يلي:

$$I\{A\} = \begin{cases} 1 & \text{if } A \text{ occurs ,} \\ 0 & \text{if } A \text{ does not occur .} \end{cases} \quad (1.5)$$

[أي يأخذ القيمة 1 إذا وقع الحدث  $A$ ، و 0 وإذا لم يقع.]

لتحدد، كمثال بسيط، عدد المرات المتوقع الذي نحصل فيه على "وجه head" عندما نرمي قطعة نقود عادلة. إن فضاء العينة هو  $S = \{H, T\}$ ، مع  $\Pr\{H\} = \Pr\{T\} = 1/2$ ، ويمكننا أن نعرف هنا المؤشر العشوائي  $X_H$  الموافق لأن تأتي الرمية بالنتيجة "وجه"، وهذا ما يعرف بالحدث  $H$ . يُقدِّم هذا المتحول عدد مرات ظهور "الوجه" في الرمية، فهو يساوي 1 إذا ظهر "الوجه"، و 0 إذا ظهر الجانب الآخر. نكتب:

$$\begin{aligned} X_H &= I\{H\} \\ &= \begin{cases} 1 & \text{if } H \text{ occurs ,} \\ 0 & \text{if } T \text{ occurs .} \end{cases} \end{aligned}$$

إن عدد الوجوه المتوقع الحصول عليه في رمية واحدة هو ببساطة القيمة المتوقعة للمؤشر  $X_H$ :

$$\begin{aligned} E\{X_H\} &= E[I\{H\}] \\ &= 1 \cdot \Pr\{H\} + 0 \cdot \Pr\{T\} \\ &= 1 \cdot (1/2) + 0 \cdot (1/2) \\ &= 1/2 . \end{aligned}$$

إذن، عدد الوجوه المتوقع الحصول عليه في رمية واحدة لقطعة نقد عادلة هو  $1/2$ . وتبين التوطئة التالية أن القيمة المتوقعة للمؤشر العشوائي الموافق لحدث  $A$  تساوي احتمال وقوع الحدث  $A$ .

### توطئة 1.5

ليكن لدينا فضاء عينة  $S$  وحدث  $A$  من فضاء العينة  $S$ ، وليكن  $X_A = I\{A\}$ ، فيكون  $E\{X_A\} = \Pr\{A\}$ .

**البرهان** اعتمادًا على تعريف للمؤشر العشوائي في المعادلة (1.5)، وعلى تعريف القيمة المتوقعة، لدينا

$$\begin{aligned} E\{X_A\} &= E[I\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\bar{A}\} \\ &= \Pr\{A\} , \end{aligned}$$

■

حيث تشير  $\bar{A}$  إلى  $A$  - متممة  $A$ .

قد يبدو من المزعج استعمال المؤشرات العشوائية في تطبيق مثل عدد المرات المتوقع فيه الحصول على

<sup>1</sup> كان من الواجب أن نستخدم التعبير "متحول عشوائي مؤشر" حرفيًا، لكننا ارتأينا - للسهولة - استخدام "مؤشر عشوائي"؛ فمن الواضح أن السياق يدل على أننا نتعامل مع متحولات عشوائية من غلط مؤشر. (للمترجم)

وجه عند رمي قطعة نقود واحدة، إلا أنها مفيدة في دراسة حالات نقوم فيها بتجارب عشوائية مكررة. فعلى سبيل المثال، تعطينا المؤشرات العشوائية طريقة بسيطة للوصول إلى نتيجة للمعادلة (ت.37). ففي هذه المعادلة نحسب عدد الوجوه عند رمي قطعة نقد  $n$  مرة، وذلك بأن ندرس على نحو منفصل احتمال الحصول على: 0 وجهًا، وجه واحد، وجهين، إلخ. إن الطريقة البسيطة المقترحة في المعادلة (ت.38) تستخدم ضمناً المؤشرات العشوائية. ولزيد من الإيضاح، بمقدرونا تسمية  $X_i$  للمؤشر العشوائي اللواقق للحدث المتمثل في الحصول على وجه في الرمية  $i$ . أي لدينا:  $X_i = I\{\text{the } i \text{ th flip results in the event } H\}$ . ليكن  $X$  المتحول العشوائي الذي يمثل عدد الوجوه الكلي في  $n$  رمية، بحيث

$$X = \sum_{i=1}^n X_i .$$

نريد أن نحسب عدد الوجوه المتوقع، لذلك فإننا نأخذ توقع طرقي للمعادلة السابقة لنحصل على

$$E[X] = E\left[\sum_{i=1}^n X_i\right] .$$

تعطي المعادلة السابقة توقع مجموع  $n$  متحولاً عشوائياً. وبالاغتماد على التوطئة 1.5، يمكننا بسهولة حساب توقع كل متحول عشوائي. وبالاغتماد على (ت.21) - خطية التوقع - يصبح حساب توقع المجموع سهلاً: فهو يساوي مجموع توقعات المتحولات العشوائية، وعددها  $n$ . إن خطية التوقع تجعل من استخدام المؤشرات العشوائية طريقة تحليلية فعالة؛ ويمكن تطبيقها حتى حين تكون للمتحولات العشوائية مرتبطة. بإمكاننا الآن أن نحسب عدد الوجوه المتوقع بسهولة:

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^n X_i\right] \\ &= \sum_{i=1}^n E[X_i] \\ &= \sum_{i=1}^n 1/2 \\ &= n/2 . \end{aligned}$$

وهكذا، ومقارنةً بالطريقة المستخدمة في المعادلة (ت.37)، نجد أن المؤشرات العشوائية تبسط الحساب كثيراً. لذا، فإننا سنستخدم المؤشرات العشوائية في كل هذا الكتاب.

### تحليل مسألة التوظيف باستخدام المؤشرات العشوائية

لنعدّ إلى مسألة التوظيف. نريد الآن أن نحسب العدد المتوقع للمرات التي نوظّف فيها موظّفاً جديداً. حتى

نتمكن من استخدام تحليل احتمالي، نفترض أن المرشحين يصلون وفق ترتيب عشوائي، كما ناقشنا في المقطع السابق. (سترى في المقطع 3.5 كيف نخذف هذه الفرضية.) ليكن  $X$  للتحول العشوائي الذي تساوي قيمته عدد مرات توظيف موظف جديد. بإمكاننا أن نطبق تعريف القيمة المتوقعة من المعادلة (ت.20) فنحصل على

$$E[X] = \sum_{x=1}^n x \Pr\{X = x\} ,$$

إلا أن هذا الحساب قد يكون مجهولاً. إذن، سنعمد بدلاً منه إلى استخدام المؤشرات العشوائية التي سنبسط الحساب كثيرًا.

ولاستخدام المؤشرات العشوائية بدلاً من حساب  $E[X]$  اعتماداً على متحول واحد موافق لعدد مرات توظيف موظف جديد، نعرف  $n$  متحولاً يتعلّق كلٌّ منها بحقيقة توظيف مرشح محدد أو لا. وبوجه خاص، نعرف  $X_i$  المؤشر العشوائي الموافق للمحدث الذي يتمّ فيه توظيف المرشح  $i$ . إذن

$$X_i = \begin{cases} 1 & \text{if candidate } i \text{ is hired} \\ 0 & \text{if candidate } i \text{ is not hired} \end{cases}$$

(أي إن  $X_i$  يأخذ القيمة 1 عندما يُوظّف المرشح  $i$ ، و 0 إذا لم يُوظّف.)

و:

$$X = X_1 + X_2 + \dots + X_n . \quad (2.5)$$

واعتماداً على التوتلة 1.5، يكون لدينا:

$$E[X_i] = \Pr\{\text{candidate } i \text{ is hired}\}$$

[أي  $E[X_i]$  تساوي احتمال توظيف المرشح  $i$ .]

علينا إذن حساب احتمال تنفيذ السطرين 6-5 من HIRE-ASSISTANT.

يُوظّف المرشح  $i$ ، في السطر 6، عندما يكون  $i$  أفضل من كل المرشحين من 1 حتى  $i-1$ . ولما افترضنا أن المرشحين يصلون وفق ترتيب عشوائي، فإن أول  $i$  مرشحاً ظهرُوا أيضاً وفق ترتيب عشوائي. واحتمال أن يكون أي واحد من هؤلاء المرشحين  $i$  الأوائل هو الأفضل متساوٍ تبعاً لمعلوماتنا. إذن احتمال أن يكون المرشح  $i$  أفضل من المرشحين من 1 إلى  $i-1$  يساوي  $1/i$ ، وهذا هو أيضاً احتمال توظيفه. واعتماداً على التوتلة 1.5، نستنتج أن

$$E[X_i] = 1/i . \quad (3.5)$$

بإمكاننا الآن حساب  $E[X]$ :

$$E[X] = E\left[\sum_{i=1}^n X_i\right] \quad \text{(من المعادلة (2.5))} \quad (4.5)$$

$$= \sum_{i=1}^n E[X_i] \quad \text{(باستخدام خطية التوقع)}$$

$$= \sum_{i=1}^n 1/i \quad \text{(من المعادلة (3.5))}$$

$$= \ln n + O(1) . \quad \text{(من المعادلة (7.1))} \quad (5.5)$$

ونحن، وإن كنا نقابل  $n$  شخصًا، فإننا في الواقع نوظف وسطيًا  $\ln n$  شخصًا منهم. نلتخص هذه النتيجة في التوظفة التالية.

### توظفة 2.5

بافتراض أن المرشحين يُردون وفق ترتيب عشوائي، فإن خوارزمية HIRE-ASSISTANT ذات كلفة توظيف كاملة من رتبة  $O(n \ln n)$ .

**البرهان** ينتج الحد مباشرة من تعريفنا لكلفة التوظيف ومن المعادلة (5.5) التي تبين أن عدد مرات التوظيف المتوقع هو تقريبًا  $\ln n$ . ■

تثل كلفة التوظيف في الحالة الوسطى تحسبًا ملموسًا مقارنة بكلفة التوظيف في أسوأ الحالات  $O(n^2)$ .

### تمارين

#### 1-2.5

ما احتمال أن توظف مرة واحدة فقط في إجراء HIRE-ASSISTANT، بافتراض أن المرشحين يُردون وفق ترتيب عشوائي؟ وما احتمال أن توظف  $n$  مرة تمامًا؟

#### 2-2.5

ما احتمال أن توظف مرتين تمامًا في إجراء HIRE-ASSISTANT، بافتراض أن المرشحين يُردون وفق ترتيب عشوائي؟

#### 3-2.5

استخدم المؤشرات العشوائية لحساب القيمة للتوقعة لمجموع  $n$  زهر نرد.

#### 4-2.5

استخدم المؤشرات العشوائية لحل المسألة التالية، التي تُعرف باسم **مسألة تعليق القبعات** *hat-check problem*: في أحد المطاعم يعطي  $n$  زبونًا قبعاتهم لموظف كي يعلقها لهم. يعيد هذا الموظف القبعات إلى



الزبائن وفق ترتيب عشوائي. ما هو العدد المتوقع للزبائن الذين يستعيدون قبعاتهم نفسها؟

### 5-2.5

لتكن  $A[1..n]$  صيغة من  $n$  عدداً متتاليًا. إذا كان  $i < j$  و  $A[i] > A[j]$ ، نقول عن الزوج  $(i, j)$  أنه قلبٌ *inversion* لـ  $A$ . (انظر المسألة 4-2 لمعرفة المزيد عن القلبات.) لنفترض أن عناصر  $A$  تشكّل تبديلاً عشوائياً منتظماً لـ  $\{1, 2, \dots, n\}$ . استخدم المؤشرات العشوائية لحساب عدد القلبات المتوقع.

## 3.5 الخوارزميات ذات العشوائية المضافة

يتينا في المقطع السابق، كيف أن معرفة تَوَزُّع المُدْخَلَات تساعدنا على تحليل سلوك خوارزمية ما في الحالة الوسطى. ولكن في كثير من الأحيان لا نمتلك هذه المعرفة، ومن ثمَّ لا يمكننا إجراء تحليل للحالة الوسطى. وقد ذكرنا في المقطع 1.5 أنه قد يكون بمقدورنا استخدام خوارزمية ذات عشوائية مضافة.

ففي مسألة كمسألة التوظيف - التي مساعدنا على تحليلها أن نفترض أن جميع التباديل على الدخول متساوية الاحتمال - سيوجهنا التحليل الاحتمالي عند بناء خوارزمية ذات عشوائية مضافة. فبدلاً من افتراض توزيع المُدْخَلَات، فإننا نفرض توزيعاً غتار. وبوجه خاص، قبل تنفيذ الخوارزمية نبذل المرشحين عشوائياً بهدف تحقيق خاصية تساوي احتمالات جميع التباديل. ومع أننا غتربنا الخوارزمية، إلا أننا ما زلنا نتوقع أن نوظف مساعداً جديداً في المكتب تقريباً  $\ln n$  مرة، ولكننا نتوقع ذلك الآن مهما كانت الدخول، بدلاً من أن يكون كذلك بافتراض مُدْخَلَات مسحوبة تبعا لتوزيع محدد.

دعنا نستكشف بعمق أكثر الفرق بين التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة. ذكرنا في المقطع 2.5، أنه بافتراض أن المرشحين يَردون تبعا لترتيب عشوائي، فإن عدد المرات المتوقع الذي يُوظَّف فيه موظفاً جديداً هي  $\ln n$ . لاحظ هنا أن الخوارزمية حتمية *deterministic*؛ فعندما نحدد دخلاً ما، سيكون عدد مرات توظيف موظف جديد هو نفسه دائماً. إضافة إلى ذلك، يختلف عدد مرات توظيف موظف جديد باختلاف المُدْخَلَات، ويتعلق بمراتب المرشحين. ولما كان هذا العدد يتعلق فقط بمراتب المرشحين، يمكننا أن نمثل أي دخل بأن تذكر مراتب المرشحين بالترتيب، أي  $(rank(1), rank(2), \dots, rank(n))$ . فإذا أُعطينا مثلاً قائمة المراتب  $A_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ ، يُوظَّف موظف جديد دوماً 10 مرات، إذ إن كل مرشح هو أفضل من سابقه. وسيتخذ السطران 5-6 في كل تكرار للخوارزمية. وإذا أُعطينا قائمة المراتب  $A_2 = \{10, 9, 8, 7, 6, 5, 4, 3, 2, 1\}$ ، فسَيُوظَّف موظف جديد مرةً واحدة فقط، في التكرار الأول. وإذا أُعطينا قائمة المراتب  $A_3 = \{5, 2, 1, 8, 4, 7, 10, 9, 3, 6\}$ ، فسَيُوظَّف موظف جديد ثلاث مرات، عند مقابلة المرشحين ذوي المراتب 5 و 8 و 10. فإذا تذكرنا أن كلفة خوارزمتنا تتعلق بعدد مرات توظيف موظف جديد، فسنجد أن هناك مُدْخَلَات مُكَلِّفة مثل  $A_1$ ، وأخرى غير مُكَلِّفة مثل  $A_2$ ، ومُدْخَلَات ذات

كلفة معتدلة مثل  $A_3$ .

لنأخذ من جهة أخرى الخوارزمية ذات العشوائية للمضافة التي تقوم أولاً بالتبديل بين المرشحين، ثم نحدد المرشح الأفضل. في هذه الحالة يكمن السلوك العشوائي داخل الخوارزمية وليس في توزيع المُدخلات. ففي دخل محدد، وليكن  $A_3$  السابق، لا نستطيع أن نحدد عدد المرات التي تُعدّل فيها القيمة العظمى، لأن هذا العدد يختلف مع كل تنفيذ للخوارزمية؛ فقد ينتج التبديل  $A_1$  في المرة الأولى التي ننفذ فيها الخوارزمية على  $A_3$ ، فتقوم الخوارزمية بـ 10 تعديلات، على حين أن التبديل  $A_2$  قد ينتج في المرة الثانية التي ننفذ فيها الخوارزمية، فنقوم بتعديل واحد فقط. وفي المرة الثالثة التي ننفذها ستقوم بعددٍ من التعديلات الأخرى. ففي كل مرة ننفذ فيها الخوارزمية، يعتمد التنفيذ على الخيارات العشوائية التي قامت بها، والتي من المحتمل أن تختلف عن التنفيذ السابق للخوارزمية. إذن فيما يخص هذه الخوارزمية والعديد من الخوارزميات الأخرى ذات العشوائية للمضافة، ليس هناك دخل محدد يتسبب في جعل الخوارزمية تسلك سلوكها في أسوأ الحالات. حتى إن أسوأ أعدالك غير قادر على توليد صيغة دخل سيئة، إذ إن التبديل العشوائي يلغي تأثير ترتيب الدخل. ولا يكون أداء الخوارزمية ذات العشوائية للمضافة سيئاً إلا إذا وُلد الأعداد العشوائية بتديلاً "سيء الحظ".

يمثل التغيير الوحيد على الرمز فيما يخص مسألة التوظيف بتبديل الصيغة عشوائياً.

#### RANDOMIZED-HIRE-ASSISTANT(n)

```

1  randomly permute the list of candidates
2  best = 0
3  for i = 1 to n
4      interview candidate i
5      if candidate i is better than candidate best
6          best = i
7      hire candidate i
```

وبهذا التغيير الطفيف، نكون قد بنينا خوارزمية ذات عشوائية مضافة يشبه أدائها أداء الخوارزمية الأصلية التي تفترض أن المرشحين يردون وفق ترتيب عشوائي.

### 3.5 توطئة

كلفة التوظيف المتوقعة للإجراء RANDOMIZED-HIRE-ASSISTANT هي  $O(c_h \ln n)$ .

**البرهان** بعد إجراء التبديل في صيغة الدخل، نكون قد وصلنا إلى حالة مماثلة لتلك التي درسناها في التحليل الاحتمالي لـ HIRE-ASSISTANT. ■

إن المقارنة بين التوطئة 2.5 والتوطئة 3.5 تُبرز الفرق بين التحليل الاحتمالي والخوارزميات ذات العشوائية المضافة. ففي التوطئة 2.5 نفترض فرضية محددة بخصوص الدخل، أما في التوطئة 3.5 فإننا لا نفترض مثل هذه

الفرضيات، إلا أن إدخال العشوائية على الدخول يستغرق زمناً إضافياً. حتى نبقى متوافقين مع مصطلحاتنا، نحددنا في التوطئة 2.5 عن زمن التوظيف في الحالة الوسطى، وفي التوطئة 3.5 عن زمن التوظيف المتوقع. سنناقش فيما تبقى من هذا المقطع بعض المسائل المتعلقة بتعديل المدخلات عشوائياً.

### تغيير الصفات عشوائياً

تُعدل العديد من الخوارزميات ذات العشوائية المضافة، العشوائية على الدخول بإجراء تغيير في صيغة الدخول المعطاة. (هناك طرق أخرى لاستخدام السلوك العشوائي.) سنناقش هنا طريقتين للقيام بذلك. نفترض أن لدينا صيغة  $A$ ، وأنها تضمّ العناصر من 1 إلى  $n$  دون أن يؤثر ذلك على العمومية. هدفنا هنا هو توليد تغيير عشوائي لعناصر الصيغة.

تتمثل إحدى الطرق الشائعة في إسناد أولوية عشوائية  $P[i]$  لكل عنصر  $A[i]$  من الصيغة، ثم نفرز عناصر  $A$  وفق هذه الأولويات. فمثلاً، إذا كانت لدينا الصيغة  $A = \{1, 2, 3, 4\}$ ، واختارنا الأولويات العشوائية  $P = \{36, 3, 62, 19\}$ ، فنسوّد الصيغة  $B = \{2, 4, 1, 3\}$ ، إذ إن الأولوية الثانية هي الصغرى، تبعها الرابعة، ثم الأولى، وأخيراً الثالثة. نسمي هذا الإجراء PERMUTE-BY-SORTING:

#### PERMUTE-BY-SORTING( $A$ )

- 1  $n = A.length$
- 2 let  $P[1..n]$  be a new array
- 3 for  $i = 1$  to  $n$
- 4      $P[i] = \text{RANDOM}(1, n^3)$
- 5 sort  $A$ , using  $P$  as sort keys

يُختار السطر 4 عدداً عشوائياً بين 1 و  $n^3$ . نستخدم ائمال من 1 إلى  $n^3$  لنجعل وحداتية الأولويات في  $P$  أمراً محتملاً. (يطلب إليك التمرين 5.3.5 أن تبرهن أن احتمال أن تكون كل الأولويات في  $P$  وحيدة هو على الأقل  $1/n - 1$ ، ويطلب إليك التمرين 6.3.5 أن تبين كيف تتخّر الخوارزمية وإن كانت أولويتان أو أكثر متماثلتين.) لنفترض أن كل الأولويات وحيدة.

الخطوة التي تستغرق وقتاً في هذا الإجراء هي الفرز في السطر 5. وسنرى في الفصل 8، أنه إذا استخدمنا فرزاً بالمقارنة، فإن الفرز يستغرق زمناً  $\Omega(n \lg n)$ . يمكننا تحقيق هذا الحد الأدنى، فقد وجدنا أن الفرز بالدمج يستغرق زمناً  $\Theta(n \lg n)$ . (ستعرّف في الباب II طرقاً أخرى للفرز بالمقارنة تستغرق  $\Theta(n \lg n)$ . يطلب إليك التمرين 3.8-4 أن تحل مسألة مشابهة جداً لفرز الأعداد في المجال بين 0 و  $n^3 - 1$  في زمن  $\Theta(n)$ .) إذا كانت  $P[i]$ ، بعد الفرز، هي الأولوية الصغرى ذات الترتيب  $i$ ، فسيكون  $A[i]$  في الموقع  $i$  من المخرج. وبهذه الطريقة نحصل على تغيير. بقي أن نبرهن أن الإجراء يؤدّي تغييراً عشوائياً منتظماً *uniform random permutation*، أي يتساوى احتمال توليد أيّ من التباديل للأعداد من 1 إلى  $n$ .

#### توطئة 4.5

يؤد الإجراء PRERMUTE-BY-SORTING تبديلاً عشوائياً منتظماً للدخل، وذلك بافتراض أن كل الأولويات متمايزة فيما بينها.

**البرهان** نبدأ بدراسة التبديل الخاص الذي يتلقى فيه كل عنصر  $A[i]$  الأولوية الصغرى ذات الترتيب  $i$ . سنبيّن أن هذا التبديل يحدث باحتمال يساوي تماماً  $1/n!$ . ليكن  $E_i$  عندما  $i = 1, 2, \dots, n$ ، الحدث المقابل لأن يتلقى العنصر  $A[i]$  الأولوية الصغرى ذات الرقم  $i$ . ونريد أن نحسب احتمال وقوع الحدث مهما كانت قيمة  $i$ ، وهو

$$\Pr\{E_1 \cap E_2 \cap E_3 \cap \dots \cap E_{n-1} \cap E_n\}.$$

اعتماداً على التمرين 5-2، يساوي هذا الاحتمال

$$\Pr\{E_1\} \cdot \Pr\{E_2|E_1\} \cdot \Pr\{E_3|E_2 \cap E_1\} \cdot \Pr\{E_4|E_3 \cap E_2 \cap E_1\} \\ \dots \Pr\{E_i|E_{i-1} \cap E_{i-2} \cap \dots \cap E_1\} \dots \Pr\{E_n|E_{n-1} \cap \dots \cap E_1\}.$$

لدينا  $\Pr\{E_1\} = 1/n$ ، لأنه احتمال أن تكون أولوية واحدة متفقا عشوائياً من بين مجموعة من  $n$  أولوية هي الصغرى. ثم نلاحظ أن  $\Pr\{E_2|E_1\} = 1/(n-1)$  لأنه يعلمنا أن  $A[1]$  أخذ الأولوية الصغرى، فإن لكل عنصر من العناصر  $n-1$  المتبقية حظاً متساوياً في أن يأخذ الأولوية الثانية في الصغر. وعموماً، عندما  $i = 2, 3, \dots, n$ ، يكون لدينا  $\Pr\{E_i|E_{i-1} \cap E_{i-2} \cap \dots \cap E_1\} = 1/(n-i+1)$ ، وذلك لأنه إذا علمنا أن العناصر من  $A[1]$  إلى  $A[i-1]$  أخذت الأولويات الصغرى، التي عددها  $i-1$  (وبالترتيب)، فإن لكل عنصر من العناصر  $n-(i-1)$  المتبقية حظاً متساوياً في أن تأخذ الأولوية الصغرى ذات الترتيب  $i$ . إذن لدينا

$$\Pr\{E_1 \cap E_2 \cap E_3 \cap \dots \cap E_{n-1} \cap E_n\} = \left(\frac{1}{n}\right) \left(\frac{1}{n-1}\right) \dots \left(\frac{1}{2}\right) \left(\frac{1}{1}\right) \\ = \frac{1}{n!},$$

وبذلك نكون قد بينا أن احتمال الحصول على التبديل المطابق هو  $1/n!$ .

يمكننا تعميم هذا البرهان على أي تبديل للأولويات. ليكن لدينا تبديل محدد  $\sigma = (\sigma(1), \sigma(2), \dots, \sigma(n))$  للمجموعة  $\{1, 2, \dots, n\}$ . ليكن  $\pi_i$  مرتبة الأولوية المستندة للعنصر  $A[i]$ ، حيث يكون للعنصر ذي الأولوية الصغرى ذات الترتيب  $i$  المرتبة  $\pi_i$ . إذا عرّفنا  $E_i$  ليكون الحدث المقابل لأن يتلقى العنصر  $A[i]$  الأولوية الصغرى ذات الترتيب  $\sigma(i)$ ، أو  $\pi_i = \sigma(i)$ ، فيمكن تطبيق البرهان السابق نفسه. إذن، إذا حسبنا احتمال الحصول على تبديل محدد، أيّا كان، فإن الحساب مماثل للحساب السابق، ويكون احتمال الحصول على هذا التبديل هو أيضاً  $1/n!$ . ■

قد يعتقد المرء أنه يكفي لبرهان أن تبديلاً ما هو تبديل عشوائي منتظم، أن نبيّن أن احتمال أن ينتهي

أي عنصر  $A[i]$  إلى الموقع  $i$  هو  $1/n$ . يبين التمرين 3.5-4 أن هذا الشرط الأضعف هو في الحقيقة غير كافٍ. هناك طريقة أفضل لتوليد تبديل عشوائي وهو تبديل الصفيفة للمطاة في المكان، حيث يقوم الإجراء RANDOMIZE-IN-PLACE بذلك في زمن  $O(n)$ . يجري في التكرار  $i$ ، اختيار العنصر  $A[i]$  عشوائيًا من بين العناصر من  $A[i]$  حتى  $A[n]$ ، ولا يطرأ أي تغيير على  $A[i]$  بعد هذا التكرار.

RANDOMIZE-IN-PLACE( $A$ )

```

1   $n = A.length$ 
2  for  $i = 1$  to  $n$ 
3      swap  $A[i]$  with  $A[RANDOM(i, n)]$ 

```

سنستخدم لامتغز حلقة لنبين أن الإجراء RANDOMIZE-IN-PLACE يولد تبديلاً عشوائيًا منتظمًا. ليكن لدينا مجموعة من  $n$  عنصرًا. نسمي المتتالية التي تضم  $k$  عنصرًا من  $n$  عنصرًا دون تكرار: تبديل- $k$  ( $k$ -permutation). (انظر الملحق ت.). هناك  $(n-k)!/n!$  تبديل- $k$  محتملاً.

### نقطة 5.5

نحسب الإجراء RANDOMIZE-IN-PLACE تبديلاً عشوائيًا منتظمًا.

**البرهان** سنستخدم لامتغز الحلقة التالي:

قبل التكرار ذي الرقم  $i$  للحلقة for في السطرين 2-3، ومهما كان التبديل- $(i-1)$  للعناصر التي عددها  $m$ ، تحتوي الصفيفة الجزئية  $A[1..i-1]$  هذا التبديل- $(i-1)$  باحتمال يساوي  $(n-i+1)/n!$ .

علينا أن نبين أن هذا الامتغز صحيح قبل التكرار الأول للحلقة، وأن كل تكرار للحلقة يحافظ على هذا الامتغز. ويجب أن نبين أيضًا أن هذا الامتغز يقدم خاصية مفيدة تسمح بالتحقق من الصحة عندما نتوقف للحلقة.

**الاستعداد:** لندرس الحالة قبل التكرار الأول للحلقة، أي  $i=1$ . إن لامتغز الحلقة يعني أنه مهما كان التبديل-0، فإن الصفيفة الجزئية  $A[1..0]$  تحتوي هذا التبديل-0 باحتمال يساوي  $1/n! = (n-i+1)/n! = 1/n!$ . الصفيفة الجزئية  $A[1..0]$  هي صفيفة جزئية فارغة، وأي تبديل-0 لا يحوي أي عنصر، إذن تحتوي الصفيفة  $A[1..0]$  أي تبديل-0، باحتمال 1، وبهذا يكون لامتغز الحلقة محققًا قبل التكرار الأول.

**المحافظة على الشرط:** نفترض أنه، قبل التكرار  $i$  مباشرة، يظهر كل تبديل- $(i-1)$  في الصفيفة الجزئية  $A[1..i-1]$  باحتمال يساوي  $(n-i+1)/n!$ ، وسوف نبين أنه بعد التكرار  $i$ ، يظهر أي تبديل- $i$

ممكّن في الصيغة الجزئية  $A[1..i]$  باحتمال  $(n-i)!/n!$ . إذن، بزيادة 1 على  $i$  للدخول في التكرار التالي سيبقى لامتغز الحلقة محققًا.

لندرس التكرار  $i$ . لنأخذ تبديل  $i$  عددًا، ولنسمّ عناصره  $\langle x_1, x_2, \dots, x_i \rangle$ . يتكوّن هذا التبديل من تبديل  $(i-1)$   $\langle x_1, x_2, \dots, x_{i-1} \rangle$  متبوع بالقيمة  $x_i$  التي تضعها الخوارزمية في  $A[i]$ . ليكن  $E_1$  الحدث المتمثل في قيام التكرارات  $i-1$  الأولى بإنشاء التبديل  $(i-1)$  المحدّد  $\langle x_1, x_2, \dots, x_{i-1} \rangle$  في  $A[1..(i-1)]$ . اعتمادًا على لامتغز الحلقة، يكون  $\Pr\{E_1\} = (n-i+1)!/n!$ . ليكن  $E_2$  الحدث المتمثل في أن يضع التكرار  $i$  العنصر  $x_i$  في الموقع  $A[i]$ . إن التبديل  $i$   $\langle x_1, \dots, x_i \rangle$  في الصيغة  $A[1..i]$  يظهر تمامًا مع حدوث كلٍّ من  $E_1$  و  $E_2$ ، ولهذا السبب نريد حساب  $\Pr\{E_2 \cap E_1\}$ . باستخدام المعادلة (ت.14)، يكون لدينا

$$\Pr\{E_2 \cap E_1\} = \Pr\{E_2|E_1\} \Pr\{E_1\}.$$

إن الاحتمال  $\Pr\{E_2|E_1\}$  يساوي  $1/(n-i+1)$  لأن السطر 3 من الخوارزمية يختار  $x_i$  عشوائيًا من بين  $n-i+1$  قيمة في المواقع  $A[i..n]$ . إذن لدينا

$$\begin{aligned} \Pr\{E_2 \cap E_1\} &= \Pr\{E_2|E_1\} \Pr\{E_1\} \\ &= \frac{1}{n-i+1} \cdot \frac{(n-i+1)!}{n!} \\ &= \frac{(n-i)!}{n!} \end{aligned}$$

الإنهاء: في النهاية،  $i = n+1$ ، ويكون لدينا أن الصيغة الجزئية  $A[1..n]$  هي تبديل  $n$  معطى باحتمال  $(n-(n+1)+1)!/n! = 0!/n! = 1/n!$ .

■ وبهذا، تنشئ RANDOMIZE-IN-PLACE تبديلاً عشوائيًا منتظمًا.

إن الخوارزمية ذات العشوائية المضافة هي في كثير من الأحيان أبسط الطرق وأكثرها فعالية لحل مسألة ما. سنستخدم الخوارزميات ذات العشوائية المضافة في مواقع عديدة في هذا الكتاب.

تمارين

1-3.5

يحتج الأستاذ مارسو Marceau على لامتغز الحلقة المستخدم في برهان التوفطة 5.5. إنه يتساءل فيما إذا كان محققًا قبل التكرار الأول. محالته مبنية على أنه بإمكان للمرء أن يصرّح ببساطة أن صيغة جزئية فارغة لا تحتوي أي تبديل 0. ولهذا السبب، فإن احتمال أن تحتوي صيغة جزئية فارغة على تبديل 0 معدوم، وهذا ما يجعل لامتغز الحلقة غير محقق قبل التكرار الأول. أعد كتابة الإجراء RANDOMIZE-IN-PLACE بحيث

يكون لامتغیر الحلقة للموافق له محققاً على صفيقة جزئية غير فارغة قبل الدخول في التكرار الأول، وعدّل برهان التوظنة 5.5 لإجرائيتك.

### 2-3.5

قرّر الأستاذ كيلب Kelp أن يكتب إجراء ينشئ عشوائياً أي تبديل باستثناء التبديل المطابق. وهو يقترح الإجراء التالي:

#### PERMUTE-WITHOUT-IDENTITY( $A$ )

```

1   $n = A.length$ 
2  for  $i = 1$  to  $n - 1$ 
3      swap  $A[i]$  with  $A[RANDOM(i + 1, n)]$ 
```

هل يقوم هذا الرماز بما ينبغيه الأستاذ كيلب؟

### 3-3.5

لفترض أنه، بدلاً من مبادلة العنصر  $A[i]$  بعنصر عشوائي من الصفيقة الجزئية  $A[i..n]$ ، فإننا نبادله بعنصر عشوائي من أي موقع في الصفيقة:

#### PERMUTE-WITH-ALL( $A$ )

```

1   $n = A.length$ 
2  for  $i = 1$  to  $n$ 
3      swap  $A[i]$  with  $A[RANDOM(1, n)]$ 
```

هل يُنتج هذا الرماز تبديلاً عشوائياً منتظماً؟ لماذا أو لا؟

### 4-3.5

يقترح الأستاذ أرمسترونغ Armstrong الإجراء التالي لتوليد تبديل عشوائي منتظم:

#### PERMUTE-BY-CYCLIC( $A$ )

```

1   $n = A.length$ 
2  let  $B[1..n]$  be a new array
3   $offset = RANDOM(1, n)$ 
4  for  $i = 1$  to  $n$ 
5       $dest = i + offset$ 
6      if  $dest > n$ 
7           $dest = dest - n$ 
8       $B[dest] = A[i]$ 
9  return  $B$ 
```

يُبين أن كل عنصر  $A[i]$  له احتمال  $1/n$  لينتهي في أي موقع محدد في  $B$ . ثم يبين أن الأستاذ أرمسترونغ مخطئ، وذلك بأن تبين أن التبديل الناتج ليس تبديلاً عشوائياً منتظماً.

### 5-3.5 \*

برهن أن احتمال أن تكون كل العناصر وحيدة في الصيغة  $P$  في الإجراء PERMUTE-BY-SORTING، هو على الأقل  $1/n - 1$ .

### 6-3.5

اشرح كيف يمكن تنجيز الخوارزمية PERMUTE-BY-SORTING لتعالج حالة تساوي أولويتين أو أكثر. أي يجب على خوارزمتك أن تُنتج تديلاً عشوائياً منتظماً ولو كانت هناك أولويتان أو أكثر متساويتين.

### 7-3.5

افترض أننا نريد إنشاء عينة عشوائية *random sample* من المجموعة  $\{1, 2, 3, \dots, n\}$ ، أي مجموعة جزئية من  $m$  عنصراً، حيث  $0 \leq m \leq n$ ، بحيث يكون احتمال إنشاء أية مجموعة من المجموعات الجزئية ذات  $m$  عنصراً متساوياً. تمثل إحدى الطرق بجعل  $A[i] = i$ ، حيث  $i = 1, 2, 3, \dots, n$ ، ثم استدعاء  $\text{RANDOMIZE-IN-PLACE}(A)$ ، وأخذ العناصر  $m$  الأولى من الصيغة. قد تقوم هذه الطريقة باستدعاء الإجراء  $\text{RANDOM}$   $n$  مرة. إذا كانت  $n$  أكبر كثيراً من  $m$ ، يمكننا إنشاء عينة عشوائية بعدد أقل من الاستدعاءات لـ  $\text{RANDOM}$ . بَيِّنْ أن الإجراء العودي التالي يعيد المجموعة الجزئية  $S$  المكونة من  $m$  عنصراً من  $\{1, 2, 3, \dots, n\}$ ، حيث تكون كل المجموعات الجزئية الممكنة متساوية الاحتمال، وذلك بالاعتماد على  $m$  استدعاء فقط لـ  $\text{RANDOM}$ .

**RANDOM-SAMPLE( $m, n$ )**

```

1  if  $i == 0$ 
2      return  $\emptyset$ 
3  else  $S = \text{RANDOM-SAMPLE}(m - 1, n - 1)$ 
4       $i = \text{RANDOM}(1, n)$ 
5      if  $i \in S$ 
6           $S = S \cup \{n\}$ 
7      else  $S = S \cup \{i\}$ 
8      return  $S$ 
```

## 4.5 \* التحليل الاحتمالي واستخدامات إضافية للمؤشرات العشوائية

يتعمق هذا المقطع المتقدم في شرح التحليل الاحتمالي عن طريق أربعة أمثلة. يحدد الأول احتمال أن يشترك شخصان من بين  $k$  شخصاً مجتمعين في غرفة يوم ميلادهما. يدرس المثال الثاني ما يحدث عندما نرمي كرات عشوائياً في سلات. ويستكشف الثالث ضربات الحظ "Streaks" المتمثلة في الحصول على وجوه متتالية عند رمي قطعة نقد. ويحلل المثال الأخير نموذجاً معدلاً لمسألة التوظيف عليك أن تتخذ فيه القرارات دون أن تقابل فعلياً كل المرشحين.



## 1.4.5 متناقضة يوم الميلاد

مثالنا الأول هو متناقضة يوم الميلاد *birthday paradox*. ما عدد الأشخاص الذين يجب أن يُوجدوا في مكان واحد حتى يكون احتمال أن يشترك اثنان منهما في يوم ميلادهما يساوي 50%؟ والجواب على عكس ما قد تتوقعه، هو عدد قليل. وهنا يكمن التناقض فالعدد في الواقع، وكما سترى الآن أقل بكثير من عدد أيام السنة أو حتى من نصف عدد أيام السنة.

سقوم، للإجابة على هذا السؤال، بالإشارة إلى الأشخاص في الغرفة بأعداد طبيعية  $k, 1, 2, \dots$ ، حيث  $k$  هو عدد الأشخاص الكلي في الغرفة. سنتعامل مسألة السنة الكبيسة ونفترض أن عدد الأيام في كل السنوات متساوي ويساوي  $n = 365$ . ليكن  $b_i$ ، حيث  $i = 1, 2, \dots, k$ ، رقم يوم ميلاد الشخص  $i$  من أيام السنة، حيث  $1 \leq b_i \leq n$ . نفترض أيضًا أن أيام الميلاد موزعة توزيعًا منتظمًا على كل أيام السنة  $n$ ، بحيث يكون  $\Pr\{b_i = r\} = 1/n$  لكل  $i = 1, 2, \dots, k$  و  $r = 1, 2, \dots, n$ .

إن احتمال أن يكون لشخصين  $i$  و  $j$  مثلاً، يوم ميلاد مشترك يتعلق باستقلال الانتقاء العشوائي لأيام الميلاد. ونفترض من الآن فصاعدًا أن أيام الميلاد مستقلة فيما بينها، وهكذا يكون احتمال أن يقع يوم ميلاد  $i$  ويوم ميلاد  $j$  معًا في اليوم  $r$  هو

$$\begin{aligned}\Pr\{b_i = r \text{ and } b_j = r\} &= \Pr\{b_i = r\} \Pr\{b_j = r\} \\ &= 1/n^2.\end{aligned}$$

ومنه، يكون احتمال أن يقع معًا في اليوم نفسه هو

$$\begin{aligned}\Pr\{b_i = b_j\} &= \sum_{r=1}^n \Pr\{b_i = r \text{ and } b_j = r\} \\ &= \sum_{r=1}^n (1/n^2) \\ &= 1/n.\end{aligned}\tag{6.5}$$

يمكننا أن نرى ببساطة أكثر، أنه ما إن يتم اختيار  $b_i$ ، فإن احتمال أن يتم اختيار  $b_j$  ليكون اليوم  $b_i$  نفسه هو  $1/n$ . إذن، احتمال أن يتماثل يوم ميلاد  $i$  و  $j$  هو نفسه احتمال أن يقع يوم ميلاد أحدهما في يوم محدد. ولكن لاحظ أنه هذه للمصادفة مرتبطة في الواقع بافتراضنا أن أيام الميلاد مستقلة فيما بينها.

يمكننا أن ندرس احتمال أن يكون، على الأقل، لاثنتين من  $k$  شخصًا، يوم الميلاد نفسه بالنظر إلى الحدث المتمم. إن احتمال أن يتماثل على الأقل اثنين من أيام الميلاد هو 1 مطروحًا منه احتمال أن تكون جميع أيام الميلاد مختلفة. إن الحدث المتمم في أن يكون لـ  $k$  شخصًا أيام ميلاد متمايزة هو

$$B_k = \bigcap_{i=1}^k A_i .$$

حيث  $A_i$  هو الحدث المتمثل في أن يكون يوم ميلاد  $i$  مختلفًا عن يوم ميلاد الشخص  $i$  لجميع قيم  $i < j$ . ولما كان بمقدورنا أن نكتب  $B_k = A_k \cap B_{k-1}$ ، فإننا نحصل من للمعادلة (ت.16) على العلاقة العودية

$$\Pr\{B_k\} = \Pr\{B_{k-1}\} \Pr\{A_k|B_{k-1}\} , \quad (7.5)$$

حيث نأخذ  $\Pr\{B_1\} = \Pr\{A_1\} = 1$  باعتباره شرطًا ابتدائيًا. وبعبارة أخرى، إن احتمال أن يكون  $b_1, b_2, \dots, b_k$  أيام ميلاد متميزة هو احتمال أن تكون  $b_1, b_2, \dots, b_{k-1}$  أيامًا متميزة مضروبًا باحتمال أن يكون  $b_i \neq b_j$  حيث  $i = 1, 2, \dots, k-1$  علمًا أن  $b_1, b_2, \dots, b_{k-1}$  متميزة.

إذا كانت الأيام  $b_1, b_2, \dots, b_{k-1}$  متميزة، فإن الاحتمال الشرطي ليكون  $b_k \neq b_i$  عندما  $i = 1, 2, \dots, k-1$  هو  $\Pr\{A_k|B_{k-1}\} = (n - k + 1)/n$ ، إذ إن  $n - (k - 1)$  لم تؤخذ من  $n$  يومًا. نطبق العلاقة العودية (7.5) تكرارًا فنحصل على

$$\begin{aligned} \Pr\{B_k\} &= \Pr\{B_{k-1}\} \Pr\{A_k|B_{k-1}\} \\ &= \Pr\{B_{k-2}\} \Pr\{A_{k-1}|B_{k-2}\} \Pr\{A_k|B_{k-1}\} \\ &\vdots \\ &= \Pr\{B_1\} \Pr\{A_2|B_1\} \Pr\{A_3|B_2\} \dots \Pr\{A_k|B_{k-1}\} \\ &= 1 \cdot \left(\frac{n-1}{n}\right) \left(\frac{n-2}{n}\right) \dots \left(\frac{n-k+1}{n}\right) \\ &= 1 \cdot \left(1 - \frac{1}{n}\right) \left(1 - \frac{2}{n}\right) \dots \left(1 - \frac{k-1}{n}\right) . \end{aligned}$$

نعطينا المراجعة (12.3)،  $1 + x \leq e^x$  :

$$\begin{aligned} \Pr\{B_k\} &\geq e^{-1/n} e^{-2/n} \dots e^{-(k-1)/n} \\ &= e^{-\sum_{i=1}^{k-1} i/n} \\ &= e^{-k(k-1)/2n} \\ &\leq 1/2 \end{aligned}$$

عندما  $-k(k-1)/2n \leq \ln(1/2)$ . إن احتمال أن تكون كل أيام الميلاد، وعددها  $k$ ، متميزة هو على الأكثر  $1/2$  عندما  $k(k-1) \geq 2n \ln 2$ ، أو بحل للمعادلة التربيعية عندما  $k \geq (1 + \sqrt{1 + (8 \ln 2)n})/2$ . ففي حالة  $n = 365$ ، يجب أن يكون لدينا  $k \geq 23$ . إذن إذا كان هناك على الأقل 23 شخصًا في غرفة، فإن احتمال أن يشترك على الأقل شخصان يوم ميلادهما هو  $1/2$  على الأقل. أما على كوكب المريخ، فإن السنة تبلغ 669 يومًا مريخيًا؛ لذلك يجب أن يكون هناك 31 مريخيًا لنحصل على الأثر نفسه.

### تحليل باستخدام المؤشرات العشوائية

يمكننا استخدام المؤشرات العشوائية لتقديم تحليل أبسط، ولكنه تقريبي، لمتناقضة يوم الميلاد. نعرف، لكل ثنائية

$(i, j)$  من  $k$  شخصاً في الغرفة، المؤشر العشوائي  $X_{ij}$ ، حيث  $1 \leq i < j \leq k$ ،

$X_{ij} = 1$  (person  $i$  and person  $j$  have the same birthday)

$$= \begin{cases} 0 & \text{if person } i \text{ and person } j \text{ have the same birthday} \\ 1 & \text{otherwise} \end{cases}$$

[أي  $X_{ij}$  يساوي 1 إذا كان الشخص  $i$  والشخص  $j$  يشتركان في يوم ميلادهما].

اعتماداً على المعادلة (6.5)، نعلم أن احتمال أن يكون لشخصين يوم الميلاد نفسه هو  $1/n$ ، إذن بالاعتماد

على التوطئة 1.5 لدينا

$E[X_{ij}] = \Pr(\text{person } i \text{ and person } j \text{ have the same birthday})$

$$= 1/n$$

إذا أخذنا  $X$  ليكون التحول العشوائي الذي يُحدُّ أزواج الأشخاص الذين يشاركون كل زوج منهم يوم ميلاد

واحد، يكون لدينا

$$X = \sum_{i=1}^k \sum_{j=i+1}^k X_{ij}$$

وبحساب التوقع للطرفين، ونطبق خطية التوقع نحصل على

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^k \sum_{j=i+1}^k X_{ij}\right] \\ &= \sum_{i=1}^k \sum_{j=i+1}^k E[X_{ij}] \\ &= \binom{k}{2} \frac{1}{n} \\ &= \frac{k(k-1)}{2n} \end{aligned}$$

إذن، عندما  $k(k-1) \geq 2n$ ، يكون العدد للتوقع لأزواج الأشخاص الذين لهم يوم ميلاد مشترك مساوياً 1

على الأقل. أي إذا كان لدينا في الغرفة على الأقل  $\sqrt{2n} + 1$  شخصاً، نتوقع أن يكون هناك على الأقل

شخصان يشتركان في يوم ميلادهما. فإذا كان  $n = 365$ ، و  $k = 28$ ، فإن العدد المتوقع لأزواج الأشخاص

الذين لهم يوم ميلاد مشترك هو  $1.0356 \approx (28 \cdot 27) / (2 \cdot 365)$ . إذن، بوجود 28 شخصاً على الأقل،

نتوقع أن يُحدُّ على الأقل زوجاً من الأشخاص الذين يشاركون في يوم ميلادهم. أما على المريخ، حيث يبلغ

طول السنة 669 يومًا مريحًا، فنحتاج على الأقل إلى 38 مريحًا.

لقد حددنا باستخدام طريقة التحليل الأولى، للتمتع على الاحتمالات فقط، عدد الأشخاص اللازم حتى يتجاوز احتمال وجود زوج يتشارك في يوم الميلاد القيع 1/2، وباستخدام طريقة التحليل الثانية، التي استخدمت المؤشرات العشوائية، حددنا العدد اللازم حتى يكون العدد المتوقع لأيام الميلاد المشتركة يساوي 1. وعلى الرغم من أن عدد الأشخاص الدقيق يختلف في الحالتين، إلا أنهما متماثلان بالمقاربة:  $O(\sqrt{n})$ .

#### 2.4.5 الكرات والسلال

سنناقش عملية الرمي العشوائي لكرات متماثلة في  $b$  سلّة مرقّمة  $1, 2, \dots, b$ ، حيث تكون الرميات مستقلة فيما بينها، ويكون احتمال أن تنتهي الكرة في أية سلّة - عند كل رمية - متساويًا. إن احتمال أن تستقر الكرة الرمية في أية سلّة محدّدة هو  $1/b$ . أي إنّ عملية رمي الكرات هي متتالية من تجارب برنولي (Bernoulli trials) (انظر الملحق ت.4). باحتمال نجاح يساوي  $1/b$ ، حيث يعني النجاح هنا أن تقع الكرة في السلّة المحدّدة. إن هذا النموذج ذو فائدة خاصة عند تحليل التليد hashing (انظر الفصل 11)، وبمكنا أن نجيب عن العديد من الأسئلة المثيرة للاهتمام بخصوص عملية رمي الكرات. (نطرح المسألة ت-1 أسئلة إضافية بشأن الكرات والسلال.)

ما عدد الكرات التي تقع في سلّة محدّدة؟ إن عدد الكرات الذي يقع في سلّة محدّدة يتبع التوزيع الثنائي الخد  $b(k; n, 1/b)$ . إذا رُميت  $n$  كرة، فإن للمعادلة (ت.37) نقرّر أن العدد المتوقع للكرات التي تقع في سلّة محدّدة هو  $n/b$ .

ما عدد الكرات التي يجب أن نرميها وسطيًا حتى تحتوي سلّة محدّدة على كرة واحدة؟ إن عدد الرميات اللازم حتى تتلقى سلّة محدّدة كرة ما يتبع التوزيع الهندسي باحتمال  $1/b$ ، واعتادًا على المعادلة (ت.32)، يكون عدد الرميات المتوقع حتى حصول النجاح هو  $1/(1/b)$ .

ما عدد الكرات الذي يجب أن نرميها حتى تحتوي كل سلّة على كرة واحدة على الأقل؟ نسّمي الرمية التي توقع كرة في سلّة فارغة "هدفًا hit". ونريد أن نعرف عدد الرميات المتوقع ■ اللازم للحصول على  $b$  هدفًا.

يمكن استخدام الأهداف لتجزئة  $n$  رمية إلى مراحل. تتكون المرحلة  $i$  من الرميات بعد الهدف  $i - 1$  وحتى الحصول على الهدف  $i$ . وتتكوّن المرحلة الأولى من رمية واحدة، إذ إنه من المؤكّد أننا سنحقّق هدفًا عندما نكون كل السلّات فارغة. عند كل رمية في المرحلة  $i$ ، يكون لدينا  $i - 1$  سلّة فيها كرات و  $i + 1 - b$  سلّة فارغة. إذن، أيّا كانت الرمية في المرحلة  $i$ ، يكون احتمال الحصول على هدف مساويًا  $(i + 1)/b$ . نسّمي  $n_i$  عدد الرميات في المرحلة  $i$ ، فيكون عدد الرميات اللازم للحصول على  $b$  هدفًا هو  $n = \sum_{i=1}^b n_i$ . إن كل متحوّل عشوائي  $n_i$  يتبع توزيعًا هندسيًا باحتمال نجاح يساوي  $(i + 1)/b$ .

واعتمادًا على المعادلة (ت.32) يكون

$$E[n_i] = \frac{b}{b-i+1}.$$

واعتمادًا على خطية التوقع، لدينا

$$\begin{aligned} E[n] &= E\left[\sum_{i=1}^b n_i\right] \\ &= \sum_{i=1}^b E[n_i] \\ &= \sum_{i=1}^b \frac{b}{b-i+1} \\ &= b \sum_{i=1}^b \frac{1}{i} \\ &= b(\ln b + O(1)). \end{aligned} \quad ((7.أ)) \text{ (اعتمادًا على للمعادلة)}$$

نستنتج مما سبق أننا بحاجة إلى  $b \ln b$  رمية تقريبًا قبل أن نتوقع أن يكون هناك كرة في كل سلة. تُعرف هذه المسألة أيضًا باسم مسألة جامع القسائم *coupon collector's problem*، التي تنص على أن الشخص الذي يحاول جمع قسيمة من كل نوع من  $b$  قسيمة مختلفة، عليه أن يخرز  $b \ln b$  قسيمة تقريبًا يحصلها عشوائيًا لكي ينجح في مسماء.

### 3.4.5 ضربات الحظ

لنفترض أنك ترمي قطعة نقد عادلة  $n$  مرة. ما هي أطول ضربة حظ streak منمتلة في سلسلة وجوه متتالية نتوقع الحصول عليها؟ الجواب هو  $\Theta(\lg n)$ ، كما يبين التحليل التالي.

نبرهن أولاً أن الطول المتوقع لأطول ضربة حظ هو  $O(\lg n)$ . إن احتمال أن يكون ناتج رمي كل قطعة وجهاً هو  $1/2$ . ليكن الحدث للمنتمل في أن ضربة حظ طولها على الأقل  $k$  تبدأ بالرمية رقم  $i$ ، أو بدقة أكبر هو الحدث للمنتمل في أن  $k$  رمية متتالية  $i, i+1, \dots, i+k-1$  تعطي وجوهاً فقط، حيث  $1 \leq k \leq n$  و  $1 \leq i \leq n-k+1$ . ولما كانت رميات قطعة النقد كلها مستقلة فيما بينها، أيًا كان الحدث للمعطى  $A_{ik}$ ، فإن احتمال أن تكون كل الرميات، وعددها  $k$ ، وجوهاً هو

$$\Pr[A_{ik}] = 1/2^k. \quad (8.5)$$

فإذا كان  $k = 2\lceil \lg n \rceil$ ، فإن

$$\begin{aligned}\Pr\{A_{i,2\lceil \lg n \rceil}\} &= 1/2^{2\lceil \lg n \rceil} \\ &\leq 1/2^{2\lg n} \\ &= 1/n^2 ,\end{aligned}$$

وبذلك يكون احتمال حدوث ضربة حظ طولها على الأقل  $2\lceil \lg n \rceil$  بدءاً من الموقع  $i$  صغيراً جداً. ولما كان عدد المواقع عندما تبدأ مثل هذه الضربة هو  $1 - 2\lceil \lg n \rceil + 1$  موفقاً على الأكثر، فإن احتمال الحصول على ضربة حظ طولها على الأقل  $2\lceil \lg n \rceil$  وجهاً تبدأ في أي موقع ممكن هو

$$\begin{aligned}\Pr\left\{\bigcup_{i=1}^{n-2\lceil \lg n \rceil+1} A_{i,2\lceil \lg n \rceil}\right\} &\leq \sum_{i=1}^{n-2\lceil \lg n \rceil+1} 1/n^2 \\ &< \sum_{i=1}^n 1/n^2 \\ &= 1/n ,\end{aligned}\tag{9.5}$$

وذلك اعتماداً على متراجحة بول (ت.19)، حيث إن احتمال اجتماع عدد من الأحداث يساوي على الأكثر مجموع احتمالات الأحداث الفردية. (لاحظ أن متراجحة بول محققة وإن لم تكن الأحداث مستقلة.) نستخدم الآن المتراجحة (9.5) لنحدّ طول أطول ضربة حظ. ليكن  $L_i$  الحدث للتمثل في أن أطول ضربة حظ طولها  $i$  تمامًا، حيث  $i = 0, 1, 2, \dots, n$ ، وليكن  $L$  طول أطول ضربة حظ. لدينا اعتماداً على تعريف القيمة المتوقعة،

$$E[L] = \sum_{j=0}^n j \Pr\{L_j\} .\tag{10.5}$$

يمكننا محاولة حساب هذا المجموع باستخدام حدود عليا على كل  $\Pr\{L_j\}$  كتلك المحسوبة في المتراجحة (9.5)، ولكن لسوء الحظ، قد تعطي هذه الطريقة حدوداً ضعيفة. إلا أنه بإمكاننا الاعتماد على بعض الحسب المكتسب من التحليل السابق لنحصل على حدّ جيد. نلاحظ، بدايةً، أنه لا توجد حدود فردية في المجموع الوارد في المعادلة (10.5) يكون فيها كلٌّ من العاملين  $j$  و  $\Pr\{L_j\}$  كبيراً. لماذا؟ لأنه عندما يكون  $j \geq 2\lceil \lg n \rceil$ ، فإن  $\Pr\{L_j\}$  يكون صغيراً جداً، وعندما يكون  $j < 2\lceil \lg n \rceil$ ، فإن قيمة  $j$  تكون صغيرة على نحو لا بأس به. وبعد دراسة أدق، نلاحظ أن الأحداث  $L_i$  منفصلة عندما  $i = 0, 1, \dots, n$ ، ومن ثمّ فإن احتمال أن تبدأ ضربة حظ طولها على الأقل  $2\lceil \lg n \rceil$  وجهاً في أي موقع هو:  $\sum_{j=2\lceil \lg n \rceil}^n \Pr\{L_j\}$ . ومن المعادلة (9.5)، يكون لدينا  $\sum_{j=2\lceil \lg n \rceil}^n \Pr\{L_j\} < 1/n$ .

وإذا لاحظنا أيضاً أن  $\sum_{j=0}^n \Pr\{L_j\} = 1$ ، يكون لدينا  $\sum_{j=0}^{2\lceil \lg n \rceil-1} \Pr\{L_j\} \leq 1$  إذن نحصل على

$$\begin{aligned}
 E[L] &= \sum_{j=0}^n j \Pr\{L_j\} \\
 &= \sum_{j=0}^{2\lceil \lg n \rceil - 1} j \Pr\{L_j\} + \sum_{j=2\lceil \lg n \rceil}^n j \Pr\{L_j\} \\
 &< \sum_{j=0}^{2\lceil \lg n \rceil - 1} (2\lceil \lg n \rceil) \Pr\{L_j\} + \sum_{j=2\lceil \lg n \rceil}^n n \Pr\{L_j\} \\
 &= 2\lceil \lg n \rceil \sum_{j=0}^{2\lceil \lg n \rceil - 1} \Pr\{L_j\} + n \sum_{j=2\lceil \lg n \rceil}^n \Pr\{L_j\} \\
 &< 2\lceil \lg n \rceil \cdot 1 + n \cdot (1/n) \\
 &< O(\lg n)
 \end{aligned}$$

إن احتمال أن تتجاوز ضربة الحظ  $r\lceil \lg n \rceil$  وجهاً تتناقص بسرعة مع تزايد  $r$ . فعندما يكون  $r \geq 1$ ، فإن احتمال أن تبدأ ضربة حظ طولها  $r\lceil \lg n \rceil$  وجهاً بدءاً من الموقع  $i$  هو

$$\begin{aligned}
 \Pr\{A_{i,r\lceil \lg n \rceil}\} &= 1/2^{r\lceil \lg n \rceil} \\
 &\leq 1/2^r.
 \end{aligned}$$

وهكذا، فإن احتمال أن يبلغ طول أطول ضربة حظ  $r\lceil \lg n \rceil$  على الأقل هو  $1/n^r = 1/n^{r-1}$  على الأكثر، أو بعبارة مكافئة: احتمال أن يكون طول أطول ضربة حظ أقل من  $r\lceil \lg n \rceil$  هو  $1 - 1/n^{r-1}$ . لنأخذ  $n = 1000$  رمية، مثلاً على ذلك، فإن احتمال الحصول على ضربة حظ طولها على الأقل  $2\lceil \lg n \rceil = 20$  وجهاً هو على الأكثر  $1/n = 1/1000$ . وحظوظ الحصول على ضربة حظ تتجاوز  $3\lceil \lg n \rceil = 30$  وجهاً هي على الأكثر  $1/n^2 = 1/1,000,000$ .

سنبرهن الآن هذا أدق متشعباً: إن الطول المتوقع لأطول ضربة حظ في  $n$  رمية هو  $\Omega(\lg n)$ . ولبرهان هذا الحد، نبحث عن ضربات حظ طولها  $s$  يتجزئ الرميات، التي عددها  $n$ ، إلى  $n/s$  مجموعة تقريباً من  $s$  رمية. فإذا اخترنا  $s = \lceil (\lg n)/2 \rceil$ ، أمكننا أن نبيّن أنه من المحتمل أن تأتي كل الرميات في إحدى هذه المجموعات وجوهاً، ونستنتج من هذا أنه من المحتمل أن تكون طول أطول ضربة حظ هو على الأقل  $s = \Omega(\lg n)$ . ونبيّن فيما بعد أن الطول المتوقع لأطول ضربة حظ هو  $\Omega(\lg n)$ .

نجزئ الـ  $n$  رمية لقطعة التقود إلى  $\lceil n/(\lg n)/2 \rceil$  مجموعة على الأقل من  $\lceil (\lg n)/2 \rceil$  رمية متتالية، ونعطي حداً لاحتمال ألا يكون هناك أية مجموعة كلها وجوه. واعتماداً على المعادلة (8.5)، يكون احتمال أن تأتي كل رميات المجموعة التي تبدأ في الموقع  $i$  وجوهاً هو

$$\Pr\{A_{i, \lfloor (\lg n)/2 \rfloor}\} = 1/2^{\lfloor (\lg n)/2 \rfloor} \\ \geq 1/\sqrt{n} .$$

إذن، احتمال ألا تبدأ ضربة حظ طولها على الأقل  $\lfloor (\lg n)/2 \rfloor$  وجهًا من الموقع  $i$  هو على الأكثر  $1 - 1/\sqrt{n}$ . ولما كانت المجموعات  $\lfloor n/\lfloor (\lg n)/2 \rfloor \rfloor$  مكونة من رميات متمايزة ومستقلة فيما بينها، فإن احتمال أن تفشل كل مجموعة من هذه المجموعات في أن تكون ضربة حظ طولها  $\lfloor (\lg n)/2 \rfloor$  هو على الأكثر

$$(1 - 1/\sqrt{n})^{\lfloor n/\lfloor (\lg n)/2 \rfloor \rfloor} \leq (1 - 1/\sqrt{n})^{n/\lfloor (\lg n)/2 \rfloor - 1} \\ \leq (1 - 1/\sqrt{n})^{2n/\lg n - 1} \\ \leq e^{-(2n/\lg n - 1)/\sqrt{n}} \\ = O(e^{-\lg n}) \\ = O(1/n) .$$

وقد استخدمنا في هذا التعليل المتراجحة (12.3)،  $1 + x \leq e^x$ ، ومتراجحة قد ترغب في التحقق منها، وهي أن  $\lg n \geq (2n/\lg n - 1)/\sqrt{n}$  عندما  $n$  كبيرة كفاية. إذن، احتمال أن تساوي أطول ضربة حظ الطول  $\lfloor (\lg n)/2 \rfloor$  أو تتجاوزها هو

$$\sum_{j=\lfloor (\lg n)/2 \rfloor + 1}^n \Pr\{L_j\} \geq 1 - O(1/n) . \quad (11.5)$$

يمكننا الآن أن نحسب حدًا أدنى على الطول المتوقع لأطول ضربة حظ، بأن نبدأ بالمعادلة (10.5)، ونقوم بالعمليات بطريقة مشابهة لما قمنا به لحساب الحد الأعلى:

$$E[L] = \sum_{j=0}^n \Pr\{L_j\} \\ = \sum_{j=0}^{\lfloor (\lg n)/2 \rfloor - 1} j \Pr\{L_j\} + \sum_{j=\lfloor (\lg n)/2 \rfloor}^n j \Pr\{L_j\} \\ \geq \sum_{j=0}^{\lfloor (\lg n)/2 \rfloor - 1} 0 \cdot \Pr\{L_j\} + \sum_{j=\lfloor (\lg n)/2 \rfloor}^n \lfloor (\lg n)/2 \rfloor \Pr\{L_j\} \\ = 0 \cdot \sum_{j=0}^{\lfloor (\lg n)/2 \rfloor - 1} \Pr\{L_j\} + \lfloor (\lg n)/2 \rfloor \sum_{j=\lfloor (\lg n)/2 \rfloor}^n \Pr\{L_j\}$$



$$\geq 0 + \lfloor (\lg n)/2 \rfloor (1 - O(1/n)) \quad ((11.5) \text{ من المتراجحة})$$

$$= \Omega(\lg n) .$$

بإمكاننا، كما كان الحال في متناقضة يوم الميلاد، الحصول على تحليل أبسط ولكنه تقريبي باستخدام المؤشرات العشوائية. ليكن  $X_{ik} = \mathbb{I}\{A_{ik}\}$  لمؤشر العشوائي المقابل لحدوث ضربة حظ طولها على الأقل  $k$  بدءًا من الرمية رقم  $i$ . لحساب العدد الكلي لمثل هذه الضربات، نعرّف

$$X = \sum_{i=1}^{n-k+1} X_{ik} .$$

وبأخذ توقعي الطرفين، وباستخدام خطيّة التوقع، يكون لدينا

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{n-k+1} X_{ik}\right] \\ &= \sum_{i=1}^{n-k+1} E[X_{ik}] \\ &= \sum_{i=1}^{n-k+1} \Pr\{A_{ik}\} \\ &= \sum_{i=1}^{n-k+1} 1/2^k \\ &= \frac{n-k+1}{2^k} . \end{aligned}$$

يمكننا، بنموذج قيم مختلفة لـ  $k$ ، حساب العدد المتوقع لضربات الحظ ذات الطول  $k$ . إذا كان هذا العدد كبيرًا (أكبر بكثير من 1)، فهذا يعني أن من المتوقع حدوث عدّة ضربات حظ طولها  $k$ ، واحتمال حدوث واحدة مرتفع. وإذا كان هذا العدد صغيرًا (أقل بكثير من 1)، فهذا يعني أن من المتوقع حدوث عدد قليل جدًا من ضربات الحظ ذات الطول  $k$ ، واحتمال حدوث ضربة حظ واحدة منخفض. إذا كان  $k = c \lg n$ ، حيث  $c$  ثابت موجب ما، نحصل على

$$\begin{aligned} E[X] &= \frac{n - c \lg n + 1}{2^{c \lg n}} \\ &= \frac{n - c \lg n + 1}{n^c} \\ &= \frac{1}{n^{c-1}} - \frac{(c \lg n - 1)/n}{n^{c-1}} \\ &= \Theta(1/n^{c-1}) . \end{aligned}$$

إذا كان  $c$  كبيراً، فإن العدد المتوقع لضربات الحظ ذات الطول  $c \lg n$  صغير جداً، ونستنتج أنه من غير المحتمل أن تحدث. من جهة أخرى، إذا كان  $c = 1/2$ ، فنحصل على  $E[X] = \Theta(1/n^{1/2-1}) = \Theta(1/n^{1/2})$ ، ونتوقع أنه سيكون هناك عدد كبير من ضربات الحظ ذات الطول  $(1/2) \lg n$ . لذا، فمن المتوقع جداً حدوث ضربة حظ واحدة من هذا الطول. بمقدورنا الآن، من هذه التوقعات الخشنة فقط، أن نستنتج أن توقع طول أطول ضربة حظ هو  $\Theta(\lg n)$ .

#### 4.4.5 مسألة التوظيف على الخط

سندرس، في هذا المثال الأخير، شكلاً معدلاً من مسألة التوظيف. لنفترض الآن أننا لا نريد أن نقابل كل المرشحين للعثور على أفضل مرشح بينهم. ولا نريد أيضاً أن نوظف ونسرح عندما تقع على مقدمين أفضل فأفضل. عوضاً عن ذلك، نريد أن نكتفي بمرشح قريب من الأفضل، وبالمقابل نريد التوظيف مرة واحدة فقط. يجب أن نحترم شرطاً واحداً للشركة: بعد كل مقابلة، يجب على الفور إما أن نعطي المكان الشاغر للمتقدم أو نرفضه. ما النسوية بين تقليل عدد المقابلات ورفع مستوى المرشح المختار قدر الإمكان؟

يمكننا أن نمذج هذه المسألة بالطريقة التالية: بمقدورنا بعد مقابلة كل متقدم، أن نعطي لكل مرشح علامة، ولتكن  $score(i)$  العلامة المحددة للمتقدم  $i$ ، ونفترض أنه لا يوجد متقدمان يحصلان على العلامة نفسها. بعد مقابلة  $i$  متقدماً، نعرف أيهم الأعلى علامة، ولكننا لا نعرف إذا كان أيٌّ من المتقدمين المتبقين  $i - n$  سيحقق علامة أعلى. نقرر اعتماد الاستراتيجية المتمثلة في اختيار عدد صحيح موجب  $n < k$ ، نقابل ونرفض أول  $k$  مرشحاً، ونوظف أول متقدم يليهم يحقق علامة أعلى من كل المرشحين السابقين. إذا تبين أن أفضل متقدم كان بين المقابلين  $k$  الأوائل، فسنوظف للمتقدم الأخير  $n$ . نصوغ هذه الاستراتيجية في الإجراء  $ON-LINE-MAXIMUM(k, n)$  الذي يعيد دليل المرشح الذي نريد أن نوظفه.

**ON-LINE-MAXIMUM( $k, n$ )**

```

1  bestscore =  $-\infty$ 
2  for  $i = 1$  to  $k$ 
3      if  $score(i) > bestscore$ 
4          bestscore =  $score(i)$ 
5  for  $i = k + 1$  to  $n$ 
6      if  $score(i) > bestscore$ 
7          return  $i$ 
8  return  $n$ 
```

نريد أن نحدد، لكل قيمة ممكنة لـ  $k$ ، احتمال أن نوظف أفضل متقدم ثم سنختار أفضل  $k$  ممكنة، وننفذ الاستراتيجية باستخدام هذه القيمة. لنفترض حالياً أن  $k$  ثابتة، ليكن  $M(j) = \max_{1 \leq i \leq j} \{score(i)\}$  أعلى علامة بين المتقدمين من 1 إلى  $j$ . ليكن  $S$  الحدث للتمثل في نجاحنا في اختيار أفضل مرشح، وليكن  $S_j$

الحدث المتمثل في نجاحنا عندما يكون أفضل مرشح هو المرشح ذا الترتيب  $i$  في المقابلات. لما كانت الأحداث المختلفة  $S_i$  منفصلة، فإن  $\Pr\{S\} = \sum_{i=1}^n \Pr\{S_i\}$ . وإذا لاحظنا أننا لا ننجح أبداً عندما يكون أفضل متقدم من بين المتقدمين  $k$  الأوائل، فيكون لدينا  $\Pr\{S_i\} = 0$  عندما  $i = 1, 2, \dots, k$ ، إذن نحصل على

$$\Pr\{S\} = \sum_{i=k+1}^n \Pr\{S_i\}. \quad (12.5)$$

سنحسب الآن  $\Pr\{S_i\}$ . حتى ننجح عندما يكون أفضل متقدم هو المتقدم  $i$ ، يجب أن يحدث أمران. أولاً، يجب أن يكون المتقدم الأفضل في الموقع  $i$ ، وهذا حدث نسبه  $B_i$ . ثانياً يجب ألا تختار الخوارزمية أي متقدم بين الموقعين  $k+1$  و  $i-1$ ، وهذا سيحدث فقط إذا وجدنا عندما يحقق  $i$  المتراجحة  $k+1 \leq j \leq i-1$  أن  $score(j) < bestscore$  في السطر 6. (ما كانت العلامات وحيدة، يمكننا تجاهل إمكان أن يكون  $score(j) = bestscore$ ). بمعنى آخر، يجب أن تكون كل العلامات  $score(k+1)$  حتى  $score(i-1)$  أصغر من  $M(k)$ ؛ إذا كان أي منها أكبر من  $M(k)$ ، فإننا سنعيد إذن دليل أول علامة تفوق  $M(k)$ . سنستخدم  $O_i$  لنشير إلى الحدث المتمثل في عدم اختيار أي من المتقدمين في المواقع من  $k+1$  حتى  $i-1$ . إن الحدثين  $B_i$  و  $O_i$  هما، ضمن الخط، مستقلان. فالحدث  $O_i$  يتعلق فقط بترتيب القيم في المواقع من  $i$  حتى  $i-1$ ، على حين أن الحدث  $B_i$  يتعلق فقط بكون القيمة في الموقع  $i$  أكبر من القيم في كل المواقع الأخرى. إن ترتيب القيم في المواقع من  $i-1$  إلى  $1$  لا يؤثر على كون القيمة في الموقع  $i$  أكبر منها كلها، والقيمة في الموقع  $i$  لا تؤثر على ترتيب القيم في المواقع من  $1$  حتى  $i-1$ . وهكذا يمكننا تطبيق المعادلة (15) لحصل على

$$\Pr\{S_i\} = \Pr\{B_i \cap O_i\} = \Pr\{B_i\} \Pr\{O_i\}.$$

من الواضح أن الاحتمال  $\Pr\{B_i\}$  هو  $1/n$ ، لأن القيمة العظمى يمكن أن تكون في أي موقع من بين  $n$  موقعاً باحتمال متساوٍ. ولكي يقع الحدث  $O_i$  يجب أن تكون القيمة العظمى للقيم في المواقع من  $i$  حتى  $i-1$  في موقع من بين المواقع  $k$  الأولى، واحتمال ورود متساوٍ في أي موقع منها بين هذه المواقع، التي عددها  $i-1$ . إذن  $\Pr\{O_i\} = k/(i-1)$  و  $\Pr\{S_i\} = k/(n(i-1))$ . وباستخدام المعادلة (12.5)، يكون لدينا

$$\begin{aligned} \Pr\{S\} &= \sum_{i=k+1}^n \Pr\{S_i\} \\ &= \sum_{i=k+1}^n \frac{k}{n(i-1)} \\ &= \frac{k}{n} \sum_{i=k+1}^n \frac{1}{i-1} \end{aligned}$$

$$= \frac{k}{n} \sum_{i=k}^{n-1} \frac{1}{i} .$$

وحتى نحدّد هذا المجموع من الأعلى ومن الأسفل، نقرّب باستخدام التكاملات. فمن المتراجحات (أ.12) يكون لدينا

$$\int_k^n \frac{1}{x} dx \leq \sum_{i=k}^{n-1} \frac{1}{i} \leq \int_{k-1}^{n-1} \frac{1}{x} dx .$$

إن حساب هذه التكاملات المعرّفة يعطينا الحدود

$$\frac{k}{n} (\ln n - \ln k) \leq \Pr\{S\} \leq \frac{k}{n} (\ln(n-1) - \ln(k-1)) .$$

وهي حدود ملاصقة نسبياً لـ  $\Pr\{S\}$ . ولما كنا نريد أن نجعل احتمال النجاح أكبر ما يمكن، فإننا سنركّز على اختيار قيمة  $k$  التي تجعل الحد الأدنى على  $\Pr\{S\}$  أكبر ما يمكن. (إلى جانب أنه من الأسهل جعل عبارة الحد الأدنى أكبر ما يمكن مقارنة بتكبير عبارة الحد الأعلى). بمفاضلة العبارة  $(k/n)(\ln n - \ln k)$  بالنسبة إلى  $k$  نحصل على

$$\frac{1}{n} (\ln n - \ln k - 1)$$

ونجعل هذا المشتق معدوماً نرى أن الحد الأدنى على الاحتمال يصبح أكبر ما يمكن عندما يكون  $\ln k = \ln n - 1 = \ln(n/e)$ ، أو بعبارة مكافئة  $k = n/e$ . إذن، إذا نفدنا استراتيجيتنا مع  $k = n/e$ ، فسنستطيع في توظيف أفضل متقدّم باحتمال قدره  $1/e$  على الأقل.

تمارين

#### 1-4.5

ما عدد الأشخاص الذين يجب أن يكونوا موجودين في غرفة حتى يكون احتمال أن يكون هناك شخص يشاركك في يوم ميلادك، هو  $1/2$  على الأقل؟ ما عدد الأشخاص اللازم حتى يكون احتمال أن يوجد شخصان على الأقل ولدا في 4 تموز، أكبر من  $1/2$ ؟

#### 2-4.5

افترض أن كرات زُميئت في  $b$  سلة. كل رمية مستقلة ومتساوى احتمال أن تنتهي أية كرة في أية سلة. ما هو عدد الرميات الكرات المتوقع قبل أن تحتوي على الأقل واحدة من السلال على كرتين؟

#### \* 3-4.5

هل من الضروري، عند دراسة متناقضة يوم الميلاد أن تكون أيام الميلاد مستقلة فيما بينها، أم أن الاستقلال الثنائي (زوجاً زوجاً) كافٍ؟ علّل إجابتك.

\* 4-4.5

كم مدعوًا يجب أن تدعو إلى حفلة حتى يصبح من المحتمل اجتماع ثلاثة أشخاص يشتركون في يوم ميلادهم؟

\* 5-4.5

ما احتمال أن تكون متتالية معرفية  $k$ -string  $k$  على مجموعة من  $n$  حرف هي فعلاً تبديل  $k$ ؟ ما علاقة هذا السؤال بمتناقضة يوم الميلاد؟

\* 6-4.5

افترض أننا رمينا  $n$  كرة في  $n$  سلة، حيث كل رمية مستقلة، واحتمال أن تنتهي الكرة في أية سلة متساوٍ أيضًا. ما هو عدد السلال الفارغة المتوقع؟ ما هو العدد المتوقع للسلال التي تحتوي كرة واحدة؟

\* 7-4.5

حسن الحظ الأدق على طول ضربة الحظ، وذلك بأن تبين أنه، عند رمي قطعة نقد عادلة  $n$  رمية، فهناك احتمال أقل من  $1/n$  ألا تحدث ضربة حظ أطول من  $\lg n - 2 \lg \lg n$  وجهاً متتاليًا.

## مسائل

### 1-5 المد الاحتمالي

بإمكاننا، باستخدام عدّاد ذي  $b$  بتاً، أن نُعدّ بالترتيب حتى  $2^b - 1$  فقط. وباستخدام **العدّ الاحتمالي probabilistic counting** الذي ابتدعه موريس R. Morris، بإمكاننا أن نُعدّ حتى قيمة أعلى بكثير مقابل خسارة بعض الدقة.

نعمل قيمة عدّاد  $i$  بمثل عدّاد  $i$  لكل  $i = 0, \dots, 2^b - 1$  حيث تشكل القيم  $n_i$  متتالية متزايدة من القيم الموجبة. نفترض أن القيمة البدائية للعدّاد هي 0، وهي تمثل عدّاد  $i = 0$ . نعمل العملية INCREMENT على عدّاد يحتوي القيمة  $i$  على نحو احتمالي. إذا كان  $i = 2^b - 1$  فسنسج تقرير خطأ فيض overflow error. وما سوى هذه الحالة، فإن العدّاد يزداد 1 باحتمال  $1/(n_{i+1} - n_i)$ ، ويبقى على حاله باحتمال  $1 - 1/(n_{i+1} - n_i)$ .

إذا اخترنا  $n_i = i$  لكل  $i \geq 0$ ، فإن العدّاد يكون عدّاداً عادياً ولكن تظهر حالات أكثر إثارة للاهتمام عندما نختار،  $n_i = 2^{i-1}$  مثلاً، عندما  $i > 0$  أو  $n_i = F_i$  (عدد فيبوناتشي ذو الرقم  $i$ ). انظر المقطع (2.3). لنفترض، في هذه المسألة، أن القيمة  $n_{i-1}$  كبيرة كفاية بحيث يكون احتمال خطأ الفيض مهملاً.

أ. بين أن القيمة للتوقعة التي يعطيها العدّاد بعد  $n$  عملية INCREMENT هي  $n$  تماماً.

ب. يعتمد تحليل تباين variance العد للمثل بالعداد على المتتالية  $n_i$ . لناخذ حالة بسيطة:  $n_i = 100i$  لكل  $i \geq 0$ . قدّر تباين القيمة للمثلة في العداد بعد  $n$  عملية INCREMENT.

## 2-5 البحث في صفيقة غير مفروزة

ندرس هذه المسألة ثلاث خوارزميات للبحث عن قيمة  $x$  في صفيقة غير مفروزة  $A$  مؤلفة من  $n$  عنصراً. لندرس الاستراتيجية ذات العشوائية للمضافة التالية: اختر دليلاً عشوائياً  $i$  في  $A$ . إذا كان  $A[i] = x$ ، ينتهي عملنا، وإلا فإننا نتابع البحث باختيار دليل عشوائي جديد في  $A$ . نتابع اختيار أدلة عشوائية من  $A$  حتى نجد دليلاً  $i$  بحيث يكون  $A[i] = x$  أو حتى نكون قد تحققنا من كل عنصر  $A$ . لاحظ أننا نختار الدليل من مجموعة الأدلة الكاملة في كل مرة، ولهذا السبب قد نتفحص عنصراً ما أكثر من مرة.

أ. اكتب شبه رماني للإجراء RANDOM-SEARCH الذي ينجز الاستراتيجية السابقة. تأكد أن إجراءك يتوقف عندما تكون كل الأدلة في  $A$  قد اختبرت.

ب. افترض أن هناك دليلاً واحداً  $i$  بحيث يكون  $A[i] = x$ . ما هو العدد المتوقع للأدلة التي يجب اختبارها في  $A$  قبل العثور على  $x$  وتوقف RANDOM-SEARCH؟

ت. بتعميم حلّك للسؤال (ب)، افترض أن هناك  $k \geq 1$  دليلاً  $i$  بحيث يكون  $A[i] = x$ . ما هو العدد المتوقع للأدلة التي يجب اختبارها في  $A$  قبل العثور على  $x$  وتوقف RANDOM-SEARCH؟ يجب أن يكون جوابك بدلالة  $n$  و  $k$ .

ث. افترض أنه لا يوجد أي دليل  $i$  بحيث يكون  $A[i] = x$ . ما هو العدد المتوقع للأدلة التي يجب اختبارها في  $A$  حتى تكون كل الأدلة في  $A$  قد اختبرت وحتى يتوقف RANDOM-SEARCH؟

لندرس الآن خوارزمية بحث عطيّة حتمية، نشر إليها DETERMINISTIC-SEARCH. تقوم الخوارزمية، تحديداً، بالبحث عن  $x$  داخل  $A$  بالترتيب، متفحصاً  $A[1], A[2], A[3], \dots, A[n]$  بالترتيب حتى نجد  $A[i] = x$  أو حتى الوصول إلى نهاية الصفيقة. افترض أن كل التباديل الممكنة لصفيقة الدخول متساوية الاحتمال.

ج. افترض أن هناك دليلاً واحداً  $i$  بحيث يكون  $A[i] = x$ . ما هو زمن التنفيذ المتوقع لـ DETERMINISTIC-SEARCH؟ وما هو زمن تنفيذ DETERMINISTIC-SEARCH في أسوأ الحالات؟

ح. بتعميم حلّك للسؤال (ج)، افترض أن هناك  $k \geq 1$  دليلاً  $i$  بحيث يكون  $A[i] = x$ . ما هو زمن التنفيذ المتوقع لـ DETERMINISTIC-SEARCH؟ وما هو زمن تنفيذ DETERMINISTIC-SEARCH في أسوأ الحالات؟ يجب أن يكون جوابك بدلالة  $n$  و  $k$ .

خ. افترض أنه لا يوجد أي دليل  $i$  بحيث يكون  $A[i] = x$ . ما هو زمن التنفيذ المتوقع لـ DETERMINISTIC-SEARCH؟ وما هو زمن تنفيذ DETERMINISTIC-SEARCH في أسوأ الحالات؟

وأخيراً، لندرس خوارزمية ذات عشوائية مضافة SCRAMBLE-SEARCH تعمل أولاً على خلط صفيقة الدخول عشوائياً، ثم على تنفيذ البحث الخطي الحتمي للعنصر هنا على الصفيقة الناتجة.

د. إذا كان  $k$  عدد الأدلة  $i$  بحيث يكون  $A[i] = x$ ، أعط زمن تنفيذ SCRAMBLE-SEARCH في أسوأ الحالات وزمن التنفيذ المتوقع في الحالتين  $k = 0$  و  $k = 1$ . عَمِّمْ حَلِّكَ لِمَعَالِجِ الحالة التي يكون فيها  $k \geq 1$ .

ذ. ما الخوارزمية التي قد تستخدمها من خوارزميات البحث الثلاث؟ اشرح جوابك.

## ملاحظات الفصل

تضم المراجع Bollobás [54]، و Hofri [174] و Spencer [321] كمّاً كبيراً من الطرق الاحتمالية المتقدمة. ويقدمُ كلٌّ من Karp [200] و Rabin [288] مناقشةً لفوائد الخوارزميات ذات العشوائية المضافة ومعاينة لها. ويقدمُ الكتاب التدريسي Motwani و Raghavan [262] دراسة موسّعة للخوارزميات ذات العشوائية المضافة.

لقد جرت دراسة عدّة نماذج معدّلة من مسألة التوظيف على نطاق واسع، والأكثر شيوعاً أن يشار إلى هذه المسائل باسم "مسائل السكرتيرة secretary problems". يُعَدُّ مقال Ajtai و Meggido و Waarts [11] مثلاً على أعمالٍ في هذا المجال.







## تمهيد

يعرض هذا الباب خوارزميات عديدة تحل مسألة الفرز التالية:

الدخل: متتالية من  $n$  عدداً  $(a_1, a_2, \dots, a_n)$ .

المخرج: تبديل (إعادة ترتيب)  $(a'_1, a'_2, \dots, a'_n)$  متتالية الدخل بحيث تحقق  $a'_1 \leq a'_2 \leq \dots \leq a'_n$ .  
تكون متتالية الدخل عادةً صيغةً من  $n$  عنصرًا، ويمكن تشغيلها بطرق أخرى مختلفة، كاللائحة المترابطة مثلاً.

### بنية المعطيات

نادراً ما تكون الأعداد التي نريد فرزها قيمًا منفصلةً عملياً، بل يكون كلٌّ منها، عادةً، جزءاً من تشكيلة من المعطيات تسمى *تسجيلة record*. تتضمن كلُّ تسجيلة *مفتاحاً key*، هو القيمة التي يجب فرزها. وتتألف بقية التسجيلة من *معطيات تابعة satellite data*، تُحرك عادةً مع المفتاح. عملياً، عندما تقوم بخوارزمية الفرز بتبديل المفاتيح، فلا بد أن تُبدل هذه الخوارزمية للمعطيات التابعة أيضاً. فإذا تضمنت كلُّ تسجيلة حجماً كبيراً من المعطيات التابعة، فإننا غالباً ما نبذل صيغةً من مؤشرات التسجيلات بدلاً من التسجيلات نفسها، وذلك لتقليص حركة للمعطيات.

إن تفاصيل التحجيز هذه هي التي تُميّز في الحقيقة خوارزميةً ما من برنامج متكامل. تُصنف خوارزمية الفرز *الطريقة method* التي تحدّد بما الترتيب للفرز، بصرف النظر عن كوننا نفرز أعداداً مستقلةً أم تسجيلات ضخمة تحوي كثيراً من بايتات المعطيات التابعة. لذلك، عندما نركز على مسألة الفرز، فإننا نفترض نموذجياً بأن الدخل مؤلّف من أعداد فقط. إن ترجمة خوارزمية فرز الأعداد إلى برنامج لفرز التسجيلات هو أمر مباشر مفاهيمياً (نظرياً)، مع أنه في حالات هندسية محدّدة، قد تجعل بعض التفاصيل الدقيقة الأخرى من مهمة البرمجة الفعلية تحدياً.

### لماذا الفرز؟

- يعتبر الكثير من علماء الحواسيب أن الفرز أهم المسائل الأساسية في دراسة الخوارزميات. وذلك لعدة أسباب:
- أحيانًا تكون الحاجة إلى فرز للبيانات أمرًا جوهريًا في تطبيق ما. فمثلاً، تحتاج المصارف عند تجهيز بيانات الزبائن إلى فرز الشيكات وفق أرقامها.
- غالبًا ما تستخدم الخوارزميات الفرز باعتباره مسألاً فرعياً أساسياً. فمثلاً، قد يكون على البرنامج الذي يرسم أغراضاً بيانية متوضعة بعضها فوق بعض، أن يفرز هذه الأغراض وفق علاقة "فوق" بحيث يمكنه رسم هذه الأغراض من الأسفل إلى الأعلى. سترى في هذا النص العديد من الخوارزميات التي تستخدم الفرز باعتباره مسألاً فرعياً.
- يمكننا أن نستنتج تشكيلة واسعة من خوارزميات الفرز، وهي تستخدم مجموعة غنية من التقنيات. والواقع، أن العديد من التقنيات الهامة المستخدمة أثناء تصميم الخوارزميات تظهر في متن خوارزميات فرز جري تطورها عبر السنين. ومن ثم، فالفرز مسألة ذات أهمية تاريخية أيضاً.
- يمكن أن نرهن وجود حد أدنى غير يدهي للفرز (كما سنفعل في الفصل 8). تطابق حدودنا العليا المثلى الحد الأدنى على نحو مقارب، وبذلك نعلم أن خوارزمتنا للفرز مثلى على نحو مقارب. إضافة إلى ذلك، يمكننا استخدام الحد الأدنى للفرز لإثبات حدود دنيا لبعض المسائل الأخرى.
- تظهر العديد من المسائل الهندسية عند تنجيز خوارزميات الفرز. قد يعتمد أسرع برنامج فرز خاص بحالة معينة على عدة عوامل، مثل للمعرفة المسبقة عن المفاتيح وللعطيات التابعة، وبنية الذاكرة (الذاكرة المخفية caches، والذاكرة الافتراضية virtual memory) للحاسوب المضيف، والبيئة البرمجية. تُعالج العديد من هذه المسائل أمثلياً على مستوى الخوارزميات، وليس بـ "إضافة تحسينات tweaking" إلى الرماز.

### خوارزميات الفرز

قدما في الفصل الثاني خوارزميتين لفرز  $n$  عدداً حقيقياً. أولاهما خوارزمية الفرز بالإدراج، وهي تتطلب زمناً قدره  $\Theta(n^2)$  في أسوأ الحالات. ولكن، لما كانت الحلقات الداخلية لهذه الخوارزمية محكمة، فهي خوارزمية فرز سريعة في المكان *in-place* في حال حجم دخل صغيرة. (تذكر أن خوارزمية فرز تفرز في المكان إذا كان عدد العناصر من صيغة الدخل - التي تخزن في أي وقت خارج الصيغة - عدداً ثابتاً). والثانية خوارزمية الفرز بالدمج، ولها زمن تنفيذ مقارب أفضل وهو  $\Theta(n \lg n)$ ، ولكن إجراء MERGE الذي تستخدمه لا يُنفَّذ في المكان.

سنقدم، في هذا الباب، خوارزميتين إضافيتين تفرزان أعداداً حقيقية لا على التعيين. الأولى خوارزمية الفرز

بالكومة Heapsort، سنعرضها في الفصل 6. نَفرز هذه الخوارزمية  $n$  عددًا في المكان في زمن  $O(n \lg n)$ ، ونُستخدم بنية معطيات هامة، تسمى كومة heap، يمكننا استخدامها أيضًا لتنحيز رتل ذو أولوية priority queue.

والثانية خوارزمية الفرز السريع quicksort، سنعرضها في الفصل 7. نَفرز هذه الخوارزمية  $n$  عددًا أيضًا في المكان، ولكن زمن تنفيذها في أسوأ الحالات هو  $\Theta(n^2)$ . ومع هذا فإن زمن تنفيذها للتوقع هو  $\Theta(n \lg n)$ ، وعادة ما يفوق أدائها عمليًا خوارزمية الفرز بالكومة. ويمتاز رماز خوارزمية الفرز السريع بأنه يحكم، كالفرز بالإدراج، لذلك فإن العامل الثابت المخفي في زمن تنفيذها صغير. وهي خوارزمية شائعة الاستخدام لفرز صفيقات دخول كبيرة.

وتُعَدُّ خوارزميات الفرز بالإدراج، والفرز بالدمج، والفرز بالكومة، والفرز السريع خوارزميات فرز بالمقارنة comparison sorts، أي إنها تحدّد الترتيب للفرز لصفة دخل بمقارنة عناصرها. يبدأ الفصل 8 بتقديم نموذج شجرة القرار decision-tree بهدف دراسة حدود أداء خوارزميات الفرز بالمقارنة. نبرهن، باستخدام هذا النموذج، وجود حد أدنى  $\Omega(n \lg n)$  لزمن تنفيذ أي فرز بالمقارنة على  $n$  دخلًا في أسوأ الحالات، موضحين بذلك أن الفرز بالكومة والفرز بالدمج هما خوارزميات فرز بالمقارنة أمثلتان على نحو مقارب.

يتابع الفصل 8 بعد ذلك ليثبت أنه يمكننا التغلب على الحد الأدنى  $\Omega(n \lg n)$  إذا استطعنا جمع معلومات عن ترتيب الدخل المفروز بأسلوب مختلف عن مقارنة العناصر. فمثلًا، نفترض خوارزمية الفرز بالعد أن أعداد الدخل تقع ضمن المجموعة  $\{0, 1, \dots, k\}$ . وباستخدام دليل الصفيقة أداة لتحديد الترتيب النسبي، نستطيع الفرز بالعد فرز  $n$  عددًا في زمن  $\Theta(k + n)$ . ومن ثم، في حال  $k = O(n)$  يكون زمن تنفيذ الفرز بالعد خطيًا بالنسبة إلى طول صفيقة الدخل. يمكن استخدام خوارزمية ذات صلة، وهي الفرز حسب الأساس radix sort، لتوسيع مجال الفرز بالعد. فإذا كان علينا فرز  $n$  عددًا صحيحًا، وكل عدد فيه  $d$  رقمًا، وكل رقم يمكن أن يأخذ قيمًا محتملة قد تصل إلى  $\lceil k \rceil$  قيمة على الأكثر، فيمكن للفرز حسب الأساس أن يفرز الأعداد في زمن  $\Theta(d(n + k))$ . وعندما يكون  $d$  ثابت و  $k$  من رتبة  $O(n)$ ، فإن الفرز حسب الأساس يُنفَّذ في زمن خطي. ثمة خوارزمية ثالثة، هي الفرز بالدلاء bucket sort، وتتطلب معرفة عن التوزيع الاحتمالي للأعداد في صفيقة الدخل. يمكن لهذه الخوارزمية أن تفرز  $n$  عددًا حقيقيًا موزعًا بانتظام uniformly في المجال نصف المفتوح  $[0, 1)$  في زمن  $O(n)$  في الحالة الوسطى.

يلخص الجدول التالي أزمان تنفيذ خوارزميات الفرز الموجودة في الفصل 2 والفصول من 6 إلى 8. تتعلّق  $n$  كالمعتاد، عدد العناصر التي ستُفرَز. ففي حالة الفرز بالعد، تكون العناصر التي ستُفرَز أعدادًا صحيحة من المجموعة  $\{0, 1, \dots, k\}$ . وفي حالة الفرز حسب الأساس، كل عنصر هو عدد من  $d$  رقمًا، حيث يأخذ كل رقم  $k$  قيمة محتملة. وفي حالة الفرز بالدلاء، نفترض أن المفاتيح هي أعداد حقيقية موزعة بانتظام ضمن المجال نصف المفتوح  $[0, 1)$ . يعطي العمود الأخير زمن التنفيذ في الحالة الوسطى أو المتوسط، مبيّنًا الحالة التي تعطيه

عندما يكون معيارًا لزمان التنفيذ في أسوأ الحالات. لم نورد زمن التنفيذ في الحالة الوسطى للفرز بالكومة لأننا لم نحله في هذا الكتاب.

الخوارزمية	زمن التنفيذ في أسوأ الحالات	زمن التنفيذ في الحالة الوسطى/المتوقع
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \lg n)$	$\Theta(n \lg n)$
Heapsort	$O(n \lg n)$	–
Quicksort	$\Theta(n^2)$	$\Theta(n \lg n)$ (المتوقع)
Counting sort	$\Theta(k + n)$	$\Theta(k + n)$
Radix sort	$\Theta(d(n + k))$	$\Theta(d(n + k))$
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (الحالة الوسطى)

### إحصائيات الترتيب

إن إحصائية الترتيب من الرتبة  $i$  لمجموعة من  $n$  عددًا هي العدد ذو الترتيب  $i$  من حيث الصغر في المجموعة. يمكن طبقًا لاختيار إحصائية الترتيب من الرتبة  $i$  بفرز الدخول وفهرسة العنصر ذي الترتيب  $i$  من الخرج. فإذا لم توجد أية فرضيات على توزيع الدخول، فإن هذه الطريقة تُنفَّذ في زمن  $\Omega(n \lg n)$ ، وهو الحد الأدنى المبرهن عليه في الفصل 8.

نبيّن في الفصل 9، أن بإمكاننا إيجاد العنصر ذي الترتيب  $i$  من حيث الصغر في زمن  $O(n)$ ، ولو كانت العناصر أعدادًا حقيقية لا على التمييز. نقدم خوارزمية ذات عشوائية مضافة بشبه رمار محكم يُنفَّذ في زمن  $\Theta(n^2)$  في أسوأ الحالات، ولكن في زمن متوقع  $O(n)$ . نقدم أيضًا خوارزمية أعقد تُنفَّذ في زمن  $O(n)$  في أسوأ الحالات.

### خلفية

مع أن أغلب هذا الباب لا يعتمد على الرياضيات المعقدة، إلا أن بعض المقاطع تتطلب دراية رياضية. وبوجه خاص، فإن تحليل الفرز السريع والفرز بالدلاء وخوارزمية إحصاء الترتيب يُستخدم الاحتمالات، التي عرضناها في الملحق ٥، إضافة إلى لمادة المتعلقة بالتحليل الاحتمالي والخوارزميات ذات العشوائية المضافة في الفصل 5. يتطلب تحليل خوارزمية إحصائيات الترتيب، الخطية الزمن في أسوأ الحالات، رياضيات أكثر تعقيدًا من الرياضيات اللازمة لتحليل أسوأ حالات لخوارزميات أخرى في هذا الباب.

## 6 الفرز بالكومة

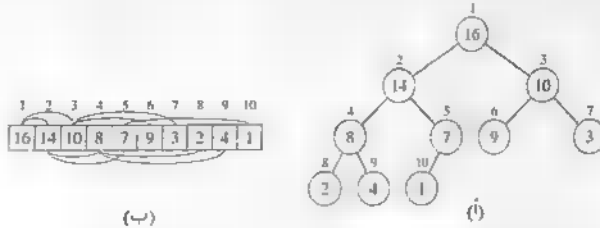
نعرض في هذا الفصل خوارزمية فرز أخرى: الفرز بالكومة `heapsort`. وزمن تنفيذ خوارزمية هذا الفرز بمائل زمن تنفيذ الفرز بالدمج، ويساوي  $O(n \lg n)$ ، وهو يخالف زمن تنفيذ الفرز بالإدراج. تفرز خوارزمية الفرز بالكومة في المكان، كالفرز بالإدراج وعلاوةً للفرز بالدمج: إذ إنه يُجرى - في أي وقت - عددًا ثابتًا فقط من عناصر الصفيفة خارج صفيفة الدخل. وهكذا، يُجمع خوارزمية الفرز بالكومة أفضل واصفات خوارزمي الفرز اللتين ناقشناهما سابقًا.

يقدم الفرز بالكومة أيضًا تقنية أخرى لتصميم الخوارزميات: وهي أنه تستخدم بنية معطيات، نسميها في هذه الحالة "كومة `heap`"، لإدارة للعلومات. ولا تقتصر الفائدة من بنية معطيات "الكومة" على الفرز بالكومة فحسب، بل تعداها لتشكيل رتل ذي أولوية فقال. ستظهر بنية المعطيات "الكومة" مجددًا في خوارزميات تُعرض في فصول لاحقة.

يصيغ المصطلح "كومة" في الأصل في سياق الفرز بالكومة، لكنه أصبح يشير إلى "تخزين النفايات المصمعة" `garbage-collected storage`، كالذي نوفره لغتا البرمجة `Lisp` و `Java`. لكن بنية المعطيات "الكومة" ليست تخزينًا للنفايات المصمعة، وحيثما نشير إلى الكومات في هذا الكتاب، فإننا نعني بنية المعطيات، وليس سمة لتجميع النفايات.

### 1.6 الكومات

بنية المعطيات **الكومة** `heap` (الثنائية `binary`) هي غرضٌ صفيفي يمكن اعتباره شجرة ثنائية كاملة تقريبًا (انظر المقطع ب-3.5)، كما هو مبين في الشكل 1-6. فكلُّ عقدة من الشجرة توافق عنصرًا من الصفيفة. وجميع مستويات الشجرة ممتلئة، ربما باستثناء للمستوى الأدنى، الذي يُملأ ابتداءً من اليسار وحتى نقطة ما. إنَّ الصفيفة `A` التي تُثَمَّل كومة ما هي غرضٌ له واصفتان: `A.length`، التي تعطي (كالعادة) عدد العناصر في الصفيفة، و `A.heap-size`، التي تمثِّل عدد عناصر الكومة المخزنة في الصفيفة `A`. أي إن - بافتراض أن `A[1..A.length]` يمكن أن تتضمن أعدادًا - العناصر الموجودة في `A[1..A.heap-size]`، حيث



**الشكل 1.6** الكومة وفق الأكبر باعتبارها (أ) شجرة ثنائية و (ب) صفيقة. إن العدد الموجود ضمن الدائرة في كل عقدة من الشجرة هو القيمة المخزنة في تلك العقدة. والعدد الموجود في أعلى عقدة ما هو الدليل الموافق في الصفيقة. تبيّن الخطوط الموجودة أسفل وأعلى الصفيقة العلاقات من غط أب-ابن؛ يكون الآباء دوماً إلى يسار أبنائهم. ارتفاع هذه الشجرة ثلاثة، وارتفاع العقدة التي دليلها 4 (وفيمتها 8) هو واحد.

يمكن  $0 \leq A.heap\text{-}size \leq A.length$ ، هي وحدها العناصر التي يمكن أن تكون في الكومة. إن حذر الشجرة هو  $A[1]$ ، وإذا أعطينا دليل عقدة ما  $i$ ، أمكننا بسهولة حساب أدلة الأب  $PARENT(i)$ ، والابن الأيسر  $LEFT(i)$ ، والابن الأيمن  $RIGHT(i)$  لهذه العقدة:

**PARENT( $i$ )**  
1 **return**  $[i/2]$

**LEFT( $i$ )**  
1 **return**  $2i$

**RIGHT( $i$ )**  
1 **return**  $2i + 1$

يمكن للإجراء **LEFT** حساب  $2i$  بتعليمة واحدة، في أغلب الحواسيب، وذلك بإزاحة تمثيل  $i$  الانشائي ببتاً واحداً نحو اليسار. وبالمثل، يمكن للإجراء **RIGHT** حساب  $2i + 1$  بسرعة، وذلك بإزاحة تمثيل  $i$  الانشائي ببتاً واحداً نحو اليمين، ثم وضع 1 في البت ذي المرتبة الدنيا. ويمكن للإجراء **PARENT** حساب  $[i/2]$  بإزاحة  $i$  ببتاً واحداً نحو اليمين. وغالباً ما تتبخر هذه الإجراءات الثلاثة، في التحيز الجيد للفرز بالكومة، باعتبارها إجراءات "ماكرو" أو إجراءات "مُضغَّنة in-line".

ثمة نوعان من الكومات الثنائية: الكومة وفق الأكبر **max-heap**، والكومة وفق الأصغر **min-heap**. وفي كليهما تحقق القيم في العقد خاصية الكومة **heap property**، أي الميزات التي يعتمد عليها نخط الكومة. **فخاصية الكومة وفق الأكبر max-heap property** هي أن كل عقدة  $i$  عدا الجذر تحقق:

$$A[PARENT(i)] \geq A[i],$$

أي إن قيمة عقدة ما تساوي على الأكثر قيمة أيها. وبذلك، يُجزئ العنصر الأكبر في كومة وفق الأكبر في الجذر، والشجرة الفرعية - التي جذرها عقدة ما - لا تتضمن قيمًا أقل أو تساوي تلك الموجودة في العقدة نفسها. أما الكومة وفق الأصغر *min-heap*، فتُنظَّم بالطريقة للمعكسة؛ فتعاصي الكومة وفق الأصغر *min-heap property* هي أنه كل عقدة  $i$  عدا الجذر تحقّق:

$$A[\text{PARENT}(i)] \leq A[i] .$$

ويكون أصغر عنصر في كومة وفق الأصغر موجودًا في جذرها.

نستخدم الكومات وفق الأكبر في خوارزمية الفرز بالكومة. تُنجزّ الكومات وفق الأصغر عادةً الأرتال ذات الأولويات، التي نناقشها في المقطع 5.6. وسنحدّد بدقة - في أيّ تطبيق معيّن - هل نحن بحاجة إلى كومة وفق الأكبر أم إلى كومة وفق الأصغر؟ فإذا كانت الخاصيات تنطبق على الكومة وفق الأكبر أو على الكومة وفق الأصغر، فإننا نستخدم للمصطلح "كومة" فقط.

بالنظر إلى الكومة على أنها شجرة، نعرّف ارتفاع *height* عقدة في كومة بأنه عدد الروصلات *edges* الموجودة على أطول مسار بسيط نازل من العقدة إلى ورقة ماء، ونعرّف ارتفاع الكومة بأنه ارتفاع جذرها. ولما كانت الكومة المولفة من  $n$  عنصرًا متبكلة على أساس شجرة ثنائية كاملة، فإن ارتفاعها هو  $\Theta(\lg n)$  (انظر التمرين 2-1.6). سنرى أن العمليات الأساسية على الكومات تُنفَّذ في زمن متناسب طرًا مع ارتفاع الشجرة على الأكثر، وبذلك فهي تستغرق زمنًا  $O(\lg n)$ . يقدّم ما تبقى من هذا الفصل عدة إجراءات أساسية ويبيّن كيفية استخدامها في خوارزمية الفرز وفي بنية للمعطيات ذات الأولوية.

- إن إجراء *MAX-HEAPIFY*، الذي يُنفَّذ في زمن  $O(\lg n)$ ، هو الأساس للمحافظة على خاصية الكومة وفق الأكبر.
- يولّد إجراء *BUILD-MAX-HEAP*، الذي يُنفَّذ في زمن خطّي، كومة وفق الأكبر من صفيقة دخل غير مرتبة.
- يُفرّز إجراء *HEAPSORT*، الذي يُنفَّذ في زمن  $O(n \lg n)$ ، صفيقة في المكان.
- تسمح الإجراءات *MAX-HEAP-INSERT* و *HEAP-EXTRACT-MAX* و *HEAP-INCREASE-KEY* و *HEAP-MAXIMUM*، التي تنفَّذ في زمن  $O(\lg n)$ ، لبنية للمعطيات بتنحيز رتل ذي أولوية.

تعاريف

1-1.6

ما هو عدد العناصر الأصغري والأعظمي في كومة ارتفاعها  $h$ ؟

2-1.6

بيّن أن ارتفاع كومة فيها  $n$  عنصرًا هو  $\lceil \lg n \rceil$ .



## 3-1.6

بيّن أنه في أية شجرة فرعية لكومة وفق الأكبر، يتضمن جذر الشجرة الفرعية القيمة العظمى للعناصر الموجودة في أي مكان في هذه الشجرة الفرعية.

## 4-1.6

أين يمكن أن يوجد أصغر عنصر في كومة وفق الأكبر، بافتراض أن جميع العناصر متمايزة؟

## 5-1.6

هل تشكل صفيحة ذات ترتيب مقرون كومة وفق الأصغر؟

## 6-1.6

هل تشكل الصفيحة التي قيمها (23, 17, 14, 6, 13, 10, 1, 5, 7, 12) كومة وفق الأكبر؟

## 7-1.6

بيّن أنه، عند استخدام التمثيل الصفيحي لفرز كومة ذات  $n$  عنصرًا، تكون الأوراق هي العقد التي دلالتها  $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \dots, n$ .

## 2.6 الحفاظ على خاصية الكومة

للمحافظة على خاصية الكومة وفق الأكبر، نستدعي الإجراء MAX-HEAPIFY. مُدخلاته هي صفيحة  $A$  ودليل  $i$  في هذه الصفيحة. يفترض MAX-HEAPIFY عند استدعائه أن الشجرتين الشائختين ذواتي الجذرين  $\text{LEFT}(i)$  و  $\text{RIGHT}(i)$  هما كومتان وفق الأكبر، ولكن قيمة  $A[i]$  يمكن أن تكون أصغر من قيمة ابنيهما، وبذلك فهي تخرق خاصية الكومة وفق الأكبر. يجعل MAX-HEAPIFY قيمة  $A[i]$  "تفوح float down" في الكومة وفق الأكبر بحيث تستجيب الشجرة الفرعية التي جذرها عند الدليل  $i$  لخاصية الكومة وفق الأكبر.

MAX-HEAPIFY( $A, i$ )

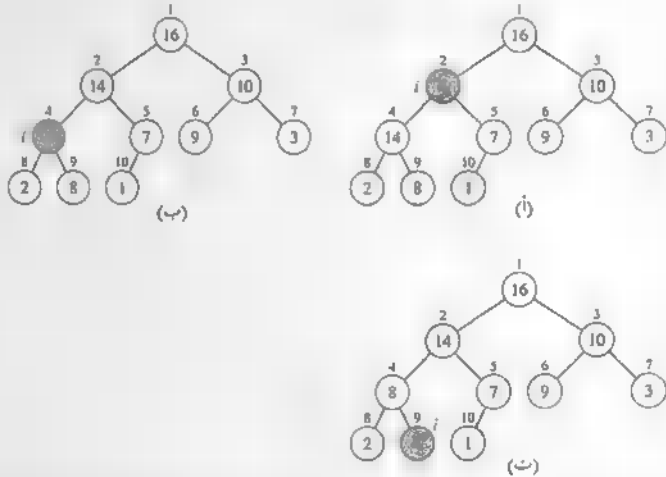
```

1  l = LEFT(i)
2  r = RIGHT(i)
3  if l ≤ A.heap-size and A[l] > A[i]
4      largest = l
5  else largest = i
6  if r ≤ A.heap-size and A[r] > A[largest]
7      largest = r
8  if largest ≠ i
9      exchange A[i] with A[largest]
10  MAX-HEAPIFY(A, largest)
```

يبيّن الشكل 2.6 كيفية عمل MAX-HEAPIFY. يجري في كل مرحلة، تحديد العنصر الأكبر لـ  $A[i]$

و  $A[\text{LEFT}(i)]$  و  $A[\text{RIGHT}(i)]$  وتخزين دليله في  $\text{largest}$ . إذا كان العنصر الأكبر هو  $A[i]$ ، تكون الشجرة الفرعية التي جذرها عند العقدة  $i$  أصلاً كومة وفق الأكبر وينتهي الإجراء. وإلا، فإن أحد الابنين يتضمن العنصر الأكبر، ونجري مبادلة  $A[i]$  مع  $A[\text{largest}]$ ، وهذا يؤدي إلى تحقيق العقدة  $i$  وابنيها خاصية الكومة وفق الأكبر. لكن الآن أصبحت العقدة التي دليها  $\text{largest}$  تحتوي القيمة الأصلية  $A[i]$ ، ومن ثم فإن الشجرة الفرعية التي جذرها  $\text{largest}$  يمكن أن تخرق خاصية الكومة وفق الأكبر. وبالنسبة، لا بد من استدعاء  $\text{MAX-HEAPIFY}$  عودياً على هذه الشجرة الفرعية.

إن زمن تنفيذ  $\text{MAX-HEAPIFY}$  على شجرة فرعية - حجمها  $n$  وجذرها عقدة معطاة  $i$  - هو الزمن  $\Theta(1)$  اللازم لتصحيح العلاقات بين العناصر  $A[i]$  و  $A[\text{LEFT}(i)]$  و  $A[\text{RIGHT}(i)]$ ، إضافة إلى الزمن اللازم لتنفيذ  $\text{MAX-HEAPIFY}$  على شجرة فرعية جذرها أحد أبناء العقدة  $i$  (بافتراض حدوث الاستدعاء العودي). حجم كل من الشجرتين الفرعيتين للابنين لا يتجاوز  $2n/3$  على الأكثر - تحدث أسوأ الحالات



**الشكل 2.6** كيفية عمل  $\text{MAX-HEAPIFY}(A, 2)$  في حال  $A.\text{heap-size} = 10$ . (أ) التشكيلة الابتدائية، حيث نخرق  $A[2]$  عند العقدة 2،  $i$  خاصية الكومة وفق الأكبر، لأنها ليست أكبر من أيٍّ من ابنيها. تُستعاد في (ب) خاصية الكومة وفق الأكبر في العقدة 2، وذلك بمبادلة  $A[2]$  بـ  $A[4]$  التي تؤدي إلى خرق خاصية الكومة وفق الأكبر للعقدة 4. لدينا الآن  $i = 4$  عند الاستدعاء العودي  $\text{MAX-HEAPIFY}(A, 4)$ . وبعد مبادلة  $A[4]$  بـ  $A[9]$ ، كما هو مبين في (ت)، يجري تصحيح العقدة 4، ولا يؤدي الاستدعاء العودي  $\text{MAX-HEAPIFY}(A, 9)$  إلى حدوث تغييرات إضافية في بنية للمعطيات.

عندما يكون نصف المستوى الأدنى من الشجرة ممثلي تماماً - ومن ثم يمكن وصف زمن تنفيذ MAX-HEAPIFY بالمعادلة التكرارية

$$T(n) \leq T(2n/3) + \Theta(1).$$

يكون حل هذه المعادلة التكرارية، حسب الحالة الثانية من النظرية العامة (النظرية 1.4)، هو  $T(n) = O(\lg n)$ . وبالمقابل، يمكننا وصف زمن تنفيذ MAX-HEAPIFY على عقدة ارتفاعها  $h$  بأنه  $O(h)$ .

تمارين

1-2.6

وضح، باستخدام الشكل 2.6 نموذجاً، كيفية تطبيق MAX-HEAPIFY( $A, 3$ ) على الصيغة

$$A = \langle 27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0 \rangle$$

2-2.6

اكتب، انطلاقاً من إجراء MAX-HEAPIFY، شبه رماز للإجراء MIN-HEAPIFY( $A, i$ ) الذي يُنجز العملية الموافقة على كومة وفق الأصغر. قارن بين زمن تنفيذ MIN-HEAPIFY وزمن تنفيذ MAX-HEAPIFY.

3-2.6

ما هي نتيجة استدعاء MAX-HEAPIFY( $A, i$ ) عندما يكون العنصر  $A[i]$  أكبر من أبنائه؟

4-2.6

ما هي نتيجة استدعاء MAX-HEAPIFY( $A, i$ ) في حالة  $i > A.\text{heap-size}/2$ ؟

5-2.6

يعتبر رماز MAX-HEAPIFY فعالاً جداً من حيث العوامل الثابتة، باستثناء الاستدعاء العودي في السطر 10، الذي يمكن أن يسبب في أن تولّد بعض المترجمات رمازاً غير فعال. اكتب إجراء MAX-HEAPIFY فعالاً نستخدم بنية تحكم تكرارية (حلقة) بدلاً من العودية.

6-2.6

يبيّن أن زمن تنفيذ MAX-HEAPIFY على كومة حجمها  $n$  في أسوأ الحالات هو  $\Omega(\lg n)$ . (نلمح: أعط، في حالة كومة ذات  $n$  عقدة، قيم عقد نوّدي إلى استدعاء عودي لـ MAX-HEAPIFY عند كل عقدة من مسار بسيط ابتداءً من الجذر نزولاً إلى ورقة.)

### 3.6 بناء كومة

يمكننا استخدام الإجراء MAX-HEAPIFY بطريقة صعودية لتحويل صيغة  $A[1..n]$ ، حيث  $n = A.\text{length}$ ، إلى كومة وفق الأكبر. استناداً إلى التمرين 1.6-7، فإن العناصر في الصيغة الجزئية

$A[(\lfloor n/2 \rfloor + 1) .. n]$  كلها أوراق الشجرة، وبذلك فكل منها يشكل كومة ذات عنصر واحد يمكن البدء به. يفحص إجراء BUILD-MAX-HEAP بقية العقد في الشجرة وينفذ MAX-HEAPIFY على كل منها.

BUILD-MAX-HEAP(A)

```
1 A.heap-size = A.length
2 for i = |A.length/2| downto 1
3   MAX-HEAPIFY(A, i)
```

يبين الشكل 3.6 مثالاً على عمل BUILD-MAX-HEAP.

ليان لماذا يعمل BUILD-MAX-HEAP بصورة صحيحة، نستخدم لامتغير الحلقة:

في بداية كل تكرار من حلقة for في الأسطر 2-3، تكون كل عقدة من العقد  $i + 1, i + 2, \dots, n$  هي الجذر لكومة وفق الأكبر.

نحتاج إلى بيان أن هذا اللامتغير صحيح قبل التكرار الأول للحلقة، وأن كل تكرار للحلقة يحافظ على اللامتغير، وأن اللامتغير يوفر خاصية مفيدة لبيان الصحة عند انتهاء الحلقة.

الاستدعاء: قبل أول تكرار للحلقة، يكون  $i = \lfloor n/2 \rfloor$ . وكل عقدة من العقد  $n, \dots, \lfloor n/2 \rfloor + 2, \lfloor n/2 \rfloor + 1$  هي ورقة، ومن ثم فمن البديهي أنها جذر كومة وفق الأكبر.

المحافظة: للتحقق من أن كل تكرار يحافظ على لامتغير الحلقة، لاحظ أن ابني العقدة  $i$  مرتان بأعداد أكبر من  $i$ . واستناداً إلى لامتغير الحلقة، فهما جذران لكومتين وفق الأكبر. وهذا هو تماماً الشرط اللازم لكي يجعل الاستدعاء  $\text{MAX-HEAPIFY}(A, i)$  العقدة  $i$  جذراً لكومة وفق الأكبر. إضافة إلى ذلك، يحافظ الاستدعاء MAX-HEAPIFY على خاصية كون جميع العقد  $n, \dots, i + 2, i + 1, i$  جذوراً لكومات وفق الأكبر. إن إنقاص  $i$  ضمن تحديث حلقة for بعيد ثمينة لامتغير الحلقة للتكرار التالي.

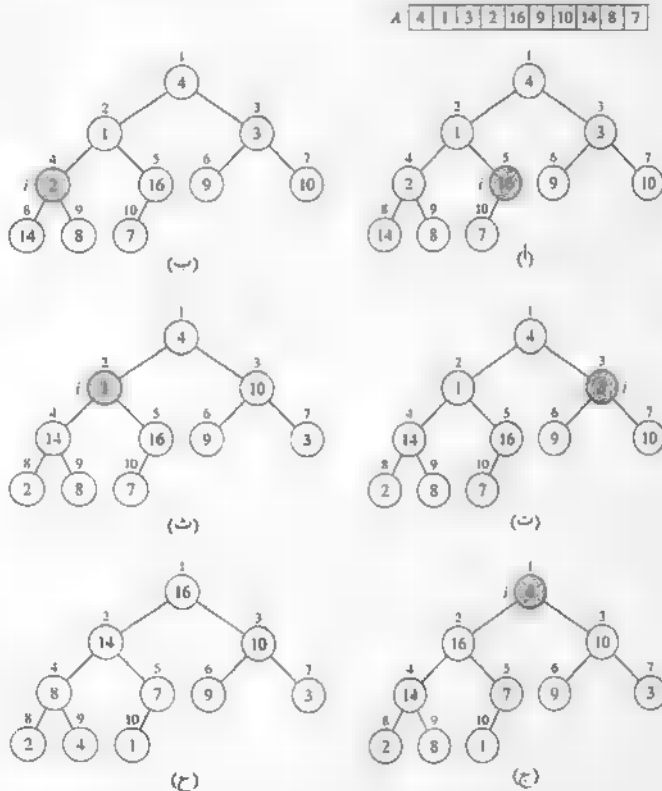
الانتهاء: لدينا عند النهاية  $i = 0$ . واستناداً إلى لامتغير الحلقة، فإن كلاً من العقد  $n, \dots, 2, 1$  هي جذر كومة وفق الأكبر. وبوجه خاص، فإن العقدة 1 هي كذلك أيضاً.

يمكننا حساب حد أعلى بسيط لزمن تنفيذ BUILD-MAX-HEAP كما يلي: يستغرق كل استدعاء لإجراء MAX-HEAPIFY زمناً  $O(\lg n)$ ، ويقوم BUILD-MAX-HEAP بـ  $O(n)$  استدعاءً مماثلاً. لذلك، فإن زمن التنفيذ هو  $O(n \lg n)$ . ومع أن هذا الحد الأعلى صحيح، إلا أنه ليس مُحكماً تقاربياً.

يمكننا استنتاج حد أكثر إحكاماً بملاحظة أن الزمن اللازم لتنفيذ MAX-HEAPIFY على عقدة يتغير بتغير ارتفاع العقدة في الشجرة، وأن ارتفاعات أغلب العقد صغيرة. يعتمد تحليلنا الأكثر إحكاماً على خاصية أن ارتفاع كومة ذات  $n$  عنصرًا هو  $\lg n$  (انظر التمرين 1.6-2)، وأن فيها  $\lceil n/2^{h+1} \rceil$  عقدة على الأكثر ارتفاعها  $h$  (انظر التمرين 3.6-3).

إن الزمن اللازم لتنفيذ MAX-HEAPIFY عند استدعائه على عقدة ارتفاعها  $h$  هو  $O(h)$ ، وبذلك يمكننا التعبير عن أن التكلفة الكلية لإجراء BUILD-MAX-HEAP محدودة من الأعلى كما يلي:

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^h} + 1 \right\rfloor O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right).$$



**الشكل 3.6** كيفية عمل BUILD-MAX-HEAP، حيث تظهر بنية المعطيات قبل استدعاء MAX-HEAPIFY في السطر 3 من BUILD-MAX-HEAP. (أ) صفيفة دخل A فيها 10 عناصر والشجرة الثنائية التي تمثلها. يبين الشكل أن دليل الحلقة 1 يشير إلى العقدة 5 قبل استدعاء MAX-HEAPIFY(A, i). (ب) بنية المعطيات الناتجة. يشير دليل الحلقة i في التكرار التالي إلى العقدة 4. (ت)-(ج) التكرارات اللاحقة للحلقة for في BUILD-MAX-HEAP. لاحظ أنه عند كل استدعاء لإجراء MAX-HEAPIFY على عقدة، تكون الشجرتان الفرعيتان لهذه العقدة كلتاهما كومة وفق الأكبر. (ح) الكومة وفق الأكبر بعد انتهاء BUILD-MAX-HEAP.

نقيم المجموع الأخير بتعويض  $x = 1/2$  في الصيغة (8.أ)، فينتج

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} \\ = 2 .$$

وبذلك، يمكن وضع حدٍّ لزم تنفيذ BUILD-MAX-HEAP كما يلي

$$O\left(n \sum_{h=0}^{\lceil \lg n \rceil} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \\ = O(n) .$$

ومن ثم، يمكننا بناء كومة وفق الأكبر انطلاقاً من صفيقة غير مرتبة في زمن خطي.

يمكننا بناء كومة وفق الأصغر باستخدام الإجراء BUILD-MIN-HEAP، الذي يشبه BUILD-MAX-HEAP مع الاستعاضة عن استدعاء MAX-HEAPIFY في السطر 3 باستدعاء MIN-HEAPIFY (انظر الترميز 2-2.6). يولد BUILD-MIN-HEAP كومة وفق الأصغر من صفيقة خطية غير مرتبة في زمن خطي.

تمارين

1-3.6

وضّح، باستخدام الشكل 3.6 نموذجاً، عمَل BUILD-MAX-HEAP على الصفيقة

$$A = (5, 3, 17, 10, 84, 19, 6, 22, 9)$$

2-3.6

لماذا نرغب في أن يتناقص دليل الحلقة  $i$  في السطر 2 من BUILD-MAX-HEAP من  $A.length/2$  إلى 1 بدلاً من أن يتزايد من 1 إلى  $A.length/2$ ؟

3-3.6

بيّن أنه يوجد على الأكثر  $\lceil n/2^{h+1} \rceil$  عقدة ارتفاعها  $h$ ، في أية كومة ذات  $n$  عنصراً.

## خوارزمية الفرز بالكومة

4.6

تبدأ خوارزمية الفرز بالكومة باستخدام BUILD-MAX-HEAP لبناء كومة وفق الأكبر من صفيقة دعل  $A[1..n]$ ، حيث  $n = A.length$ . ولما كان العنصر الأكبر في الصفيقة مخزناً في الجذر  $A[1]$ ، فيمكن وضعه في موقعه النهائي الصحيح بتبديله بـ  $A[n]$ . إذا تجاهلنا الآن العقدة  $n$  من الكومة – ويمكننا فُتْل ذلك ببساطة بتقليص  $A.heap-size$  – نلاحظ أن أبناء الجذر تبقى كوماتٍ وفق الأكبر، ولكن يمكن لعنصر

الجذر الجديد أن يترك خاصية الكومة وفق الأكبر. ومع ذلك، فإن كل ما نحتاج إليه لاستعادة خاصية الكومة وفق الأكبر هو استدعاء إجراء  $\text{MAX-HEAPIFY}(A, 1)$ ، الذي يُبقي في  $A[1 \dots n-1]$  كومة وفق الأكبر. بعد ذلك، نعيد دوارزمية الفرز بالكومة هذه الإجرائية على الكومة وفق الأكبر التي حجمها  $n-1$  لتصل إلى كومة حجمها 2. (انظر التمرين 2-4.6 لتحديد لامتغير الحلقة.)

**HEAPSORT(A)**

```

1  BUILD-MAX-HEAP(A)
2  for  $A.i = A.length$  downto 2
3      exchange  $A[1]$  with  $A[i]$ 
4       $A.heap-size = A.heap-size - 1$ 
5      MAX-HEAPIFY(A, 1)
```

يعرض الشكل 4-6 مثلاً لتطبيق HEAPSORT بعد أن يقوم السطر 1 ببناء الكومة وفق الأكبر الأولية. ويبين هذا الشكل الكومة وفق الأكبر قبل إجراء أول تكرار لحلقة for في الأسطر 2-5 وبعد كل تكرار. تستغرق إجرائية HEAPSORT زمناً  $O(n \lg n)$ ، لأن استدعاء BUILD-MAX-HEAP يستغرق زمناً  $O(n)$ ، ويستغرق كل من الـ  $n-1$  استدعاء لإجراء MAX-HEAPIFY زمناً  $O(\lg n)$ .

تمارين

1-4.6

وضّح، باستخدام الشكل 4.6 نموذجاً، عمّل HEAPSORT على الصيغة

$A = \langle 5, 13, 2, 25, 7, 17, 20, 8, 4 \rangle$

2-4.6

برهن صحة HEAPSORT باستخدام لامتغير الحلقة التالي:

في بداية كل تكرار لحلقة for في الأسطر 2-5، تكون الصيغة الجزئية  $A[1 \dots i]$  كومة وفق الأكبر حاوية أصغر  $i$  عنصراً من  $A[1 \dots n]$ ، وتتضمن الصيغة الجزئية  $A[i+1 \dots n]$  أكبر  $n-i$  عنصراً من  $A[1 \dots n]$  مرتبة.

3-4.6

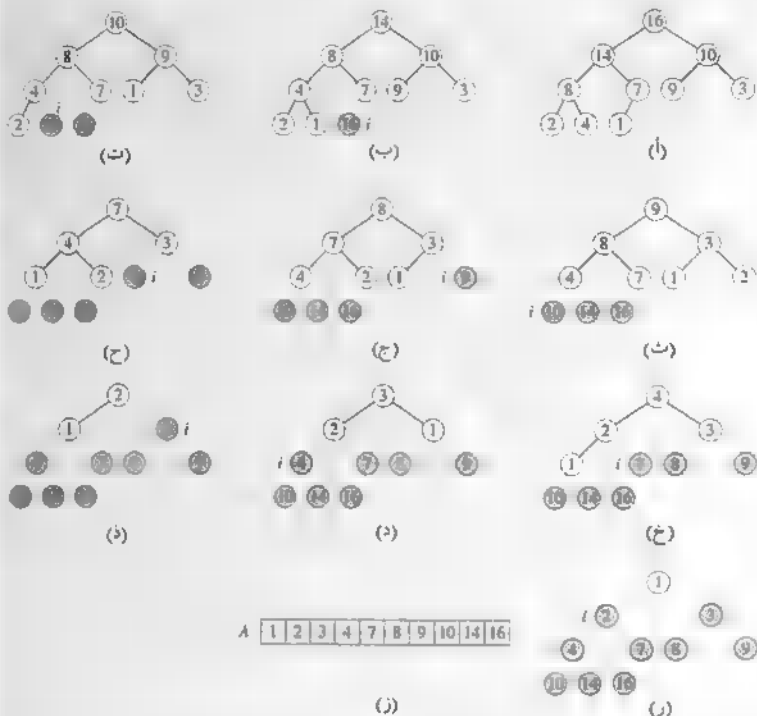
ما زمن تنفيذ الفرز بالكومة على صيغة  $A$  طولها  $n$  مرتبة أصلاً ترتيباً متزايداً؟ ماذا عن الترتيب المتناقص؟

4-4.6

بين أن زمن تنفيذ HEAPSORT في أسوأ الحالات هو  $\Omega(n \lg n)$ .

\* 5-4.6

بين أنه عندما تكون جميع العناصر متمايزة، يكون زمن تنفيذ HEAPSORT بأحسن الحالات هو  $\Omega(n \lg n)$ .



**الشكل 4.6** كيفية عمل الفرز بالكومة. (أ) بنية للمعطيات الكومة وفق الأكبر مباشرة بعد أن بناها BUILD-MAX-HEAP في السطر 1. (ب)-(و) الكومة وفق الأكبر مباشرة بعد كل استدعاء لإجراء MAX-HEAPIFY في السطر 5، حيث تُعرض قيمة  $i$  في ذلك الوقت. يبقى في الكومة العقد المظلمة نظلياً خفيماً فقط. (ز) الصفيفة A المفروزة الناتجة.

## 5.6 الأرتال ذات الأولوية

يُعدُّ الفرز بالكومة خوارزمية ممتازة، غير أننا ستعرض تحيزاً جيداً للفرز السريع في الفصل 7 يتفوق عليها عملياً في العادة. ومع ذلك، فإن لبنية للمعطيات "الكومة" نفسها استخدامات عديدة. نعرض، في هذا المقطع، إحدى أكثر تطبيقات الكومة شيوعاً، مثل: رتل أولويات فعال. وكما في الكومات، هناك نوعان من الأرتال ذات الأولويات: الأرتال ذات أولوية الأكبر max-priority queues، والأرتال ذات أولوية الأصغر min-priority queues. سنركز هنا على كيفية تنحيز الأرتال ذات أولوية الأكبر والتي تعتمد بدورها على



الكومات وفق الأكبر، يُطلَب إليك في التمرين 3-5.6 أن تكتب إجراءات الأرتال ذات أولوية الأصغر.

**الرتل ذو الأولوية priority queue** هو بنية معطيات تسمح بشخزين مجموعة  $S$  من العناصر، يرفق مع كل منها قيمة تسمى **مفتاحاً key**. يدعم **الرتل ذو أولوية الأكبر max-priority queue** العمليات التالية:

INSERT( $S, x$ ): تُدرج العنصر  $x$  في المجموعة  $S$ ، وهذا يكافئ العملية  $S = S \cup \{x\}$ .

MAXIMUM( $S$ ): تعيد العنصر ذا أكبر مفتاح من  $S$ .

EXTRACT-MAX( $S$ ): تُحذف العنصر ذا أكبر مفتاح من  $S$  وتعيده.

INCREASE-KEY( $S, x, k$ ): تزيّد قيمة مفتاح العنصر  $x$  إلى القيمة الجديدة  $k$ ، التي يفترض أن تكون مساوية على الأقل قيمة مفتاح  $x$  الحالي.

يمكننا استخدام الأرتال ذات أولوية الأكبر، من بين تطبيقاتها الأخرى المتعددة، في جدولة المهام على حاسوب مشترك. يحتفظ الرتل ذو أولوية الأكبر بمسار المهام التي يجب إنجازها وأولوياتها الموافقة. عند انتهاء مهمة أو انقطاعها، يختار المجدول scheduler المهمة ذات الأولوية العليا من بين المهام المتعلقة باستدعاء EXTRACT-MAX. يمكن للمجدول أن يضيف مهمة جديدة إلى الرتل في أي وقت باستدعاء INSERT.

وبالمقابل، يدعم الرتل ذو أولوية الأصغر min-priority queue العمليات INSERT و EXTRACT-MIN و DECREASE-KEY. يمكن استخدام الرتل ذي أولوية الأصغر في محاكاة مُقوِّد بالأحداث event-driven simulator. فالعناصر في الرتل هي أحداث يجب محاكاتها، ويُرفق مع كل منها زمن الحدوث الذي يُستخدم باعتباره مفتاحاً لها. ويجب محاكاة الأحداث وفق ترتيب زمن حدوثها، لأن محاكاة حدث ما يمكن أن يسبب محاكاة أحداث أخرى في المستقبل. يستدعي برنامج المحاكاة إجراء EXTRACT-MIN في كل مرحلة لاختيار الحدث التالي الذي يجب محاكاته. كلما تولدت أحداث جديدة، يضيفها المجدول إلى الرتل ذي أولوية الأصغر باستدعاء INSERT. سنرى في الفصولين 23 و 24 استخدامات أخرى للأرتال ذات أولوية الأصغر تركز على عملية DECREASE-KEY.

من غير المستغرب أن يكون بإمكاننا استخدام الكومة لتحيز رتل ذي أولوية. ففي تطبيق ما مثل جدولة المهام، أو محاكاة مُقوِّد بالأحداث، توافق عناصر الرتل ذي الأولوية أغراضاً objects في ذلك التطبيق. وغالباً ما نحتاج إلى تحديد غرض التطبيق الذي يوافق عنصر الرتل ذي الأولوية، والعكس بالعكس. لذلك فإننا غالباً ما نحتاج - عند استخدام كومة لتحيز رتل ذي أولوية - إلى تخزين مقيض handle يشير إلى غرض التطبيق الموافق في كل عنصر من الكومة. تعتمد بنية للمقيض الفعلية (مثل مؤشر أو عدد صحيح) على التطبيق. وبالمثل، نحتاج إلى تخزين مقيض يشير إلى عنصر الكومة الموافق في كل غرض من التطبيق. في هذه الحالة، يمكن أن يكون للمقيض دليل صفيفة array index. عند التنجيز الفعلي، وبسبب تغيير عناصر الكومة لمواقعها ضمن الصفيفة خلال العمليات على الكومة، يجب - عند تغيير موقع عنصر في الكومة - تحديث (تعديل)

دليل الصفيقة في غرض التطبيق للموافق. ولما كانت تفاصيل النفاذ إلى أغراض التطبيقات تعتمد كثيراً على التطبيق وعلى تنجيذه، فلن نتابع فيها هنا، بل سنؤكد فقط أنه لا بد من المحافظة - عملياً - على هذه المقايض بطريقة صحيحة.

نناقش فيما يلي كيفية تنجيز عمليات الرتل ذي أولوية الأصغر. ينجُز الإجراء **HEAP-MAXIMUM** عملية **MAXIMUM** في زمن  $\Theta(1)$ .

**HEAP-MAXIMUM(A)**

```
1 return A[1]
```

وينجُز الإجراء **HEAP-EXTRACT-MAX** عملية **EXTRACT-MAX**، وهو شبيهة بحسم حلقة **for** (الأسطر 3-5) من الإجراء **HEAPSORT**.

**HEAP-EXTRACT-MAX(A)**

```
1 if A.heap-size < 1
2   error "heap underflow"
3 max = A[1]
4 A[1] = A[A.heap-size]
5 A.heap-size = A.heap-size - 1
6 MAX-HEAPIFY(A, 1)
7 return max
```

إن زمن تنفيذ **HEAP-EXTRACT-MAX** هو  $O(\lg n)$ ، لأنه يقوم بإنجاز عمل ثابت فقط زيادةً على زمن **MAX-HEAPIFY** الذي هو  $O(\lg n)$ .

ينجُز الإجراء **HEAP-INCREASE-KEY** عملية **INCREASE-KEY**. يُجَدُّ دليل  $i$  في الصفيقة عنصر الرتل ذي الأولوية الذي نرغب في زيادة قيمة مفتاحه. يعدّل الإجراء أولاً قيمة مفتاح العنصر  $A[i]$  إلى قيمته الجديدة. وحيث إن زيادة قيمة مفتاح  $A[i]$  قد تُخرق خاصية الكومة وفق الأكبر، فإن الإجراء **HEAP-INCREASE-KEY** يجرّ (بطريقة مشابهة لحلقة الإدراج (في الأسطر 5-7) من **INSERTION-SORT** المذكورة في المقطع 1.2) مساراً بسيطاً ابتداءً من هذه العقدة باتجاه الجذر لإيجاد المكان المناسب للمفتاح الذي جرت زيادة قيمته. ويقارن **HEAP-INCREASE-KEY** - خلال عبوره هذا المسار - عنصراً ما بآييه بصورة تكرارية، ويبادل بين مفتاحيهما، فيتابع إذا كان مفتاح العنصر أكبر، وينتهي إذا كان مفتاح العنصر أصغر، وذلك لأن خاصية الكومة وفق الأكبر أصبحت محققة الآن. (انظر التمرين 5-5.6 للتلعق بـ لامتغير الحلقة الدقيق.)

**HEAP-INCREASE-KEY(A, i, key)**

```
1 if key < A[i]
2   error "new key is smaller than current key"
3 A[i] = key
```

```

4 while  $i > 1$  and  $A[\text{PARENT}(i)] < A[i]$ 
5     exchange  $A[i]$  with  $A[\text{PARENT}(i)]$ 
6      $i = \text{PARENT}(i)$ 

```

يُظهر الشكل 5.6 مثلاً على كيفية عمل HEAP-INCREASE-KEY. إن زمن تنفيذ HEAP-INCREASE-KEY على كومة ذات  $n$  عنصراً هو  $O(\lg n)$ ، لأن طول المسار للرسم من العقدة المُعدَّلة في السطر 3 وحتى الجذر هو  $O(\lg n)$ .

يُنجز الإجراء MAX-HEAP-INSERT عملية INSERT. إن دخل هذا الإجراء هو مفتاح العنصر الجديد الذي سيُدخَل في الكومة وفق الأكبر  $A$ . يوسَّع الإجراء أولاً الكومة وفق الأكبر بإضافة ورقة جديدة مفتاحها  $-\infty$  إلى الشجرة. بعد ذلك، يستدعي HEAP-INCREASE-KEY لوضع القيمة الصحيحة في مفتاح العقدة الجديدة، وللحفاظ على خاصية الكومة وفق الأكبر.

MAX-HEAP-INSERT( $A, key$ )

```

1  $A.heap\text{-}size = A.heap\text{-}size + 1$ 
2  $A[A.heap\text{-}size] = -\infty$ 
3 HEAP-INCREASE-KEY( $A, A.heap\text{-}size, key$ )

```

إن زمن تنفيذ MAX-HEAP-INSERT على كومة ذات  $n$  عنصراً هو  $O(\lg n)$ . وبالمجمل، فإن الكومة يمكنها أن تدعم أية عملية خاصة بالأرتال ذات الأولوية على مجموعة مؤلَّفة من  $n$  عنصراً في زمن  $O(\lg n)$ .

تمارين

1-5.6

اشرح كيفية عمل HEAP-EXTRACT-MAX على الكومة  $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$ .

2-5.6

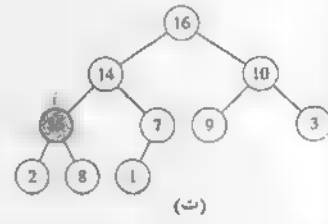
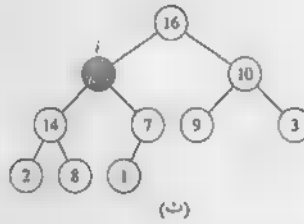
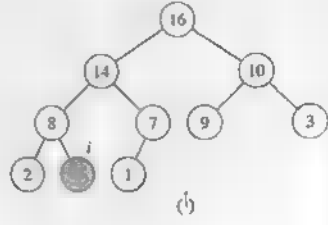
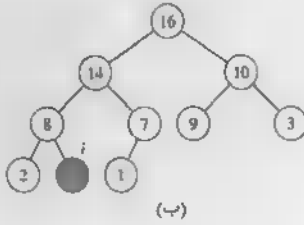
اشرح كيفية عمل MAX-HEAP-INSERT( $A, 10$ ) على الكومة  $A = \langle 15, 13, 9, 5, 12, 8, 7, 4, 0, 6, 2, 1 \rangle$ .

3-5.6

اكتب شبه رماز للإجراءات التالية: HEAP-MINIMUM و HEAP-EXTRACT-MIN و HEAP-DECREASE-KEY و MIN-HEAP-INSERT، التي تنجز رتلاً ذا أولوية الأصغر باستخدام كومة وفق الأصغر.

4-5.6

لماذا نغيب أنفسنا بوضع القيمة  $-\infty$  في مفتاح العقدة الواجب إضافتها إلى السطر 2 من MAX-HEAP-INSERT عندما تكون العملية التالية التي سننفذها بعدها مباشرة هي زيادة مفتاح هذه العقدة إلى القيمة المطلوبة؟



الشكل 5.6 كمنية عمل HEAP-INCREASE-KEY. (أ) الكومة وفق الأكبر الموجودة في الشكل 4.6(أ) وقد طُلِّت فيها العقدة ذات الدليل  $i$  تظليلاً شديداً. (ب) زُيِّدَت قيمة مفتاح هذه العقدة إلى 15. (ت) بعد تكرار واحد لحلقة `while` الموجودة في الأسطر 4-6، تبادلت العقدة وأبوها مفتاحيهما، وانتقل الدليل  $i$  صعوداً إلى الأب. (ث) الكومة وفق الأكبر بعد تكرار آخر لحقة `while`. لدينا الآن  $A[i] \geq A[\text{PARENT}(i)]$ ، وأصبحت خاصية الكومة وفق الأكبر محققة الآن، وينتهي الإجراء.

### 5-5.6

برهن صحة HEAP-INCREASE-KEY باستخدام لاستغير الحلقة التالي:

عند بداية كل تكرار لحقة `while` في الأسطر 4-6 لدينا  $A[\text{PARENT}(i)] \geq A[\text{LEFT}(i)]$  و  $A[\text{PARENT}(i)] \geq A[\text{RIGHT}(i)]$ ، إذا كانت هذه العقد موجودة وكانت الصفيفة الجزئية  $A[1..A.\text{heap-size}]$  تحقق خاصية الكومة وفق الأكبر، باستثناء إمكان حصول خرق واحد: أن يكون  $A[i]$  أكبر من  $A[\text{PARENT}(i)]$ .

يمكنك افتراض أن الصفيفة الجزئية  $A[1..A.\text{heap-size}]$  تحقق خاصية الكومة وفق الأكبر عند استدعاء HEAP-INCREASE-KEY.

### 6-5.6

تتطلب كل عملية تبديل في السطر 5 من HEAP-INCREASE-KEY نموذجاً ثلاثاً إسنادات assignments.

يُبين كيف نستخدم فكرة الحلقة الداخلية في INSERTION-SORT لتقليص هذه الإسنادات الثلاثة إلى إسناد واحد فقط.

### 7-5.6

يُبين كيف يمكن تخزين رتل "الداخل أولاً، خارج أولاً" باستخدام رتل ذي أولوية. ويُبين كيف يمكن تخزين مكسب stack باستخدام رتل ذي أولوية. (عرّفنا الأرتال والمكدسات في المقطع 1.10).

### 8-5.6

تُحدف العملية  $\text{HEAP-DELETE}(A, i)$  العنصر الموجود في العقدة  $i$  من الكومة  $A$ . أعطِ تحجيراً لإجراء  $\text{HEAP-DELETE}$  زمن تنفيذ على كومة وفق الأكبر ذات  $\Theta(\lg n)$ .

### 9-5.6

أعطِ خوارزمية زمن تنفيذها  $O(n \lg k)$  تدمج  $k$  لائحة مرتبة في لائحة مرتبة واحدة، حيث  $n$  هو العدد الكلي للعناصر في جميع لوائح الدخل. (نسمح: استخدم كومة وفق الأصغر لدمج  $k$  لائحة)

## مسائل

### 1-6 بناء كومة باستخدام الإدراج

يمكننا بناء كومة بتكرار استدعاء  $\text{MAX-HEAP-INSERT}$  لإدراج العناصر في الكومة. لنأخذ النسخة المعدلة التالية من الإجراء  $\text{BUILD-MAX-HEAP}$ :

```

BUILD-MAX-HEAP'(A)
1  A.heap-size = 1
2  for l = 2 to A.length
3      MAX-HEAP-INSERT(A, A[l])

```

أ. هل ينشئ الإجراء  $\text{BUILD-MAX-HEAP}$  و  $\text{BUILD-MAX-HEAP}'$  الكومة نفسها دونما عند تنفيذها على صيغة الدخل نفسها؟ برهن أن هذا صحيح، أو أعطِ مثالاً معاكساً.

ب. يُبين أن  $\text{BUILD-MAX-HEAP}'$  يتطلب في أسوأ الحالات زمناً  $\Theta(n \lg n)$  لبناء كومة ذات  $n$  عنصراً.

### 2-6 تحليل الكومات ذات $d$ فرعاً

الكومة ذات  $d$  فرعاً  $d$ -ary heap تشبه الكومة الثنائية (باستثناء اختلاف وحيث محتمل) وهو أن العقد للفايزة للأوراق لها  $d$  أبناً بدلاً من اثنين.

أ. كيف يمكنك تمثيل كومة ذات  $d$  فرعاً باستخدام صيغة؟

- ب. ما هو ارتفاع كومة ذات  $d$  فرعاً و  $n$  عنصراً بدلالة  $n$  و  $d$ ؟
- ت. أعطِ تنجيماً فعالاً لإجراء EXTRACT-MAX في كومة وفق الأكبر ذات  $d$  فرعاً. حلّل زمن تنفيذها بدلالة  $d$  و  $n$ .
- ث. أعطِ تنجيماً فعالاً لإجراء INSERT في كومة وفق الأكبر ذات  $d$  فرعاً. حلّل زمن تنفيذها بدلالة  $d$  و  $n$ .
- ج. أعطِ تنجيماً فعالاً لإجراء INCREASE-KEY( $A, i, k$ ) الذي يعطي رسالة خطأ في حالة  $k < A[i]$ ، وفي الحالة المعاكسة يُفُذ الاستناد  $A[i] = k$ ، ثم يُعَدّل بنية الكومة وفق الأكبر ذات  $d$  فرعاً بصورة ملائمة. حلّل زمن تنفيذ هذا الإجراء بدلالة  $d$  و  $n$ .

### 3-6 جداول يونغ

**جدول يونغ** Young tableau هو مصفوفة  $n \times m$  بحيث أن عناصر كل سطر مرتبة من اليسار إلى اليمين، وعناصر كل عمود مرتبة من الأعلى إلى الأسفل. يمكن لبعض عناصر جدول يونغ أن تكون  $\infty$ ، التي سنعاملها على أنها عناصر غير موجودة. لذلك، يمكن استخدام جدول يونغ لتخزين  $r$  عدداً منتهياً حيث أن  $r \leq mn$ .

- أ. ارسم جدول يونغ  $4 \times 4$  يتضمن العناصر  $\{9, 16, 3, 2, 4, 8, 5, 14, 12\}$ .
- ب. برهن أن جدول يونغ  $Y$  الذي بعده  $m \times n$  يكون خالياً إذا كان  $Y[1,1] = \infty$ . وأن  $Y$  يكون ممثلاً (أي يتضمن  $mn$  عنصراً) إذا كان  $Y[m,n] < \infty$ .
- ت. اكتب خوارزمية لتخزين EXTRACT-MIN على جدول يونغ  $m \times n$  غير خالي، بحيث تُنفذ في زمن  $O(m+n)$ . يجب أن تستخدم خوارزمتك مساقاً فرعياً عودياً بحلّ مسألة  $m \times n$  محلّ مسألة فرعية بعدها  $(m-1) \times n$  أو  $m \times (n-1)$  عودياً (تلميح: فكر بالإجراء MAX-HEAPIFY). عرّف  $T(p)$ ، حيث  $p = m+n$ ، ليكون زمن التنفيذ الأعظم لإجراء EXTRACT-MIN على أي جدول يونغ  $m \times n$ . اكتب معادلة تكرارية للزمن  $T(p)$  تحقق الحدّ الزمني  $O(m+n)$ ، وحلّها.

- ث. بيّن كيفية إدراج عنصر جديد في جدول يونغ  $m \times n$  غير ممتلئ في زمن  $O(m+n)$ .
- ج. بيّن، دون استخدام أية طريقة فرز أخرى كمساق فرعي، كيفية استخدام جدول يونغ بعده  $n \times n$  لفرز  $n^2$  عدداً في زمن  $O(n^3)$ .
- ح. اكتب خوارزمية تُنفذ في زمن  $O(m+n)$  لتحديد كون عددٍ ما محزّناً في جدول يونغ معطى، بعده  $m \times n$ .

## ملاحظات الفصل

اينكر Williams [357] خوارزمية الفرز بالكومة، ووصف كذلك كيفية تنجيز رتل ذي أولوية باستخدام كومة. واقترح Floyd في [106] الإجراء BUILD-MAX-HEAP.

نستخدم في الفصول 16 و 23 و 24 الكومات وفق الأصغر لتنجيز الأرتال ذات أولوية الأصغر. ونقدم في الفصل 19 تنجيزًا ذا حدود زمنية محسنة لعمليات معينة، وفي الفصل 20 افترض بأن المفاتيح مشتقة من مجموعة محدودة من أعداد صحيحة غير سالبة.

يبن Fredman و Willard في [115] كيفية تنجيز MINIMUM في زمن  $O(1)$ ، وكيفية تنجيز INSERT و EXTRACT-MIN في زمن  $O(\sqrt{\lg n})$ ، وذلك في حال كانت المعطيات أعدادًا صحيحة ذات  $b$  بتًا، وكانت ذاكرة الحاسوب مؤلفة من كلمات ذات  $b$  بتًا قابلة للبتونة. وحسن Thorup في [337] الحد  $O(\sqrt{\lg n})$  ليصبح  $O(\lg n)$ . نستخدم هذا الحد حين نخرين غير محدود بالقيمة  $n$ ، ولكن يمكن تنجيزها بحجم خطي باستخدام تقطيع ذي عشوائية مضافة randomized hashing.

نحدث حالة خاصة هامة من الأرتال ذات الأولوية عندما تكون متتالية العمليات EXTRACT-MIN *مطردة monotone*، أي إن القيم التي تعيدها عمليات EXTRACT-MIN المتتالية تتزايد باطراد مع الزمن. تظهر هذه الحالة في العديد من التطبيقات الهامة، مثل خوارزمية Dijkstra لحساب أقصر مسار من منبع واحد، التي نتاقشها في الفصل 24، وفي محاكاة الأحداث المتقطعة discrete-event simulation. من الضروري جدًا في خوارزمية Dijkstra أن يجري، على وجه الخصوص، تنجيز عملية DECREASE-KEY بغاعلية. في حالة الاطراد وكون المعطيات أعدادًا صحيحة ضمن المجال  $1, 2, \dots, C$ ، وصف Ahuja و Tarjan و Orlin و Mehlhorn في [8] كيفية تنجيز INSERT و EXTRACT-MIN في زمن محدد  $O(\lg C)$  (لمزيد من المعلومات عن التحليل المحدث انظر الفصل 17)، وكيفية تنجيز DECREASE-KEY في زمن  $O(1)$ ، وذلك باستخدام بنية معطيات تسمى كومة الأسس radix heap. يمكن تحسين الحد  $O(\lg C)$  ليصبح  $O(\sqrt{\lg C})$  باستخدام كومات فيبوناتشي Fibonacci (انظر الفصل 19) إضافة إلى كومات الأسس. وأدخل Cherkassky و Goldberg و Silverstein في [66] تحسينًا إضافيًا على هذا الحد ليصل إلى زمن متوقع  $O(\lg^{1/3+\epsilon} C)$ ، وذلك بدمج بنية الدلاء المتعددة لمستويات التي ابتكرها Denardo و Fox في [86] مع كومة Thorup المذكورة آنفًا. وأدخل Raman في [291] تحسينًا آخر على هذه النتائج لنصل إلى  $O(\min(\lg^{1/4+\epsilon} C, \lg^{1/3+\epsilon} n))$  في حالة أي  $\epsilon > 0$  محدد.

الفرز السريع خوارزمية زمن تنفيذها في حالة صفيقة دخل فيها  $n$  عددًا هو  $\Theta(n^2)$  في أسوأ الحالات. وغالبًا ما يُعدُّ الفرز السريع، على الرغم من بطء زمن تنفيذه في أسوأ الحالات، أفضل خيار عملي للفرز، لأنه فعال جدًا في الحالة العامة: فزمن تنفيذه المتوقع هو  $\Theta(n \lg n)$ ، والعوامل الثابتة المنخفضة في تدوين  $\Theta(n \lg n)$  صغيرة جدًا. وهذه الخوارزمية أيضًا ميزة الفرز في المكان (انظر المقطع 1.2)، وهي تعمل جيدًا حتى في بيئات الذاكرة الافتراضية virtual memory.

يُصف المقطع 1.7 الخوارزمية ومساقًا فرعيًا هامًا يستخدمه الفرز السريع في عملية التجزئة partitioning. ونظرًا لتعقيد سلوك خوارزمية الفرز السريع، سنبدأ بمناقشة بدئية لأدائها في المقطع 2.7 ونحلل تحليلها الدقيق إلى نهاية هذا الفصل. نعرض المقطع 3.7 نسخة من الفرز السريع تستعمل عُنَابَ عشوائية. وهذه الخوارزمية زمن تنفيذ متوقع جيد، ولا يستخلص دخل خاص سلوكها في أسوأ الحالات. يحلّل المقطع 4.7 الخوارزمية ذات العشوائية المضافة randomized algorithm، حيث يبرهن أنها تُنفَّذُ في زمن  $\Theta(n^2)$  في أسوأ الحالات، وفي زمن متوقع  $O(n \lg n)$  عند افتراض أن جميع عناصر الصفيقة متمايزة.

## 1.7 وصف الفرز السريع

يعتمد الفرز السريع، مثل الفرز بالمرج merge sort، مبدأ "فرق تسد" المذكور في المقطع 1.3-2. نعرض فيما يلي إجرائية "فرق تسد" ذات المراحل الثلاث لفرز صفيقة جزئية نموذجية  $A[p..r]$ :

**فَرِّقْ:** جَزِّئْ (أَعِدْ تَرْتِيبَ) الصفيقة  $A[p..r]$  إلى صفيقتين جزئيتين (يمكن أن تكونا خاليتين)  $A[p..q-1]$  و  $A[q+1..r]$  بحيث يكون كل عنصر من  $A[p..q-1]$  أصغر من  $A[q]$  أو يساويه، والذي هو بدوره، أصغر من أي عنصر من عناصر  $A[q+1..r]$  أو يساويه. احسب الدليل  $q$  باعتباره جزءًا من إجراء التجزئة هذا.

**سُدْ:** افِرز الصفيقتين الجزئيتين  $A[p..q-1]$  و  $A[q+1..r]$  باستدعاء عَوْدِي للفرز السريع.



ادمج: لما كانت الصفيقتان الجزئيتان مفروقتين سلفًا، فلا داعي لدمجهما: فالصفيغة  $A[p..r]$  كلها مفروزة الآن.

ينجُز الإجراء التالي الفرز السريع.

QUICKSORT( $A, p, r$ )

```

1  if  $p < r$ 
2       $q = \text{PARTITION}(A, p, r)$ 
3      QUICKSORT( $A, p, q - 1$ )
4      QUICKSORT( $A, q + 1, r$ )
    
```

لفرز كامل الصفيغة  $A$ ، يكون الاستدعاء البدلي من الشكل QUICKSORT( $A, 1, A.length$ ).

### تجزئة الصفيغة

يُعدُّ إجراء PARTITION، الذي يعيد ترتيب الصفيغة الجزئية  $A[p..r]$  في المكان، أساسَ خوارزمية الفرز السريع.

PARTITION( $A, p, r$ )

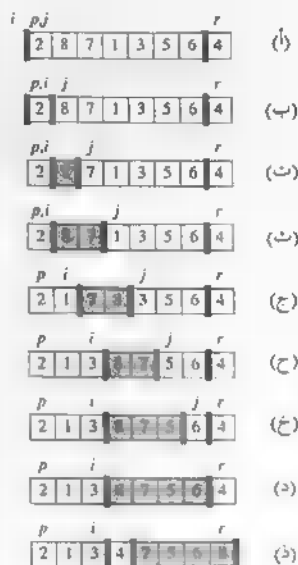
```

1   $x = A[r]$ 
2   $i = p - 1$ 
3  for  $j = p$  to  $r - 1$ 
4      if  $A[j] \leq x$ 
5           $i = i + 1$ 
6      exchange  $A[i]$  with  $A[j]$ 
7  exchange  $A[i + 1]$  with  $A[r]$ 
8  return  $i + 1$ 
    
```

يبين الشكل 1.7 كيف يعمل PARTITION على صفيغة من 8 عناصر. يختار PARTITION دائمًا عنصرًا  $x = A[r]$  ليكون عنصرًا محوريًا *pivot* تجزئ الصفيغة الجزئية  $A[p..r]$  بالنسبة إليه. يُجرى الإجراء، عند تشغيله، الصفيغة إلى أربع مناطق (يمكن أن تكون خالية). عند بداية كل تكرار من حلقة for في السطور 3-6، تحقق للمناطق خواصَّ محدَّدة مبينة في الشكل 2.7. نعتبر هذه الخواص لامتغير الحلقة loop invariant:

لدينا في بداية كل تكرار من الحلقة في السطور 3-6، ولكل دليل  $k$  من الصفيغة:

1. إذا كان  $i \leq k \leq p$  فإن  $A[k] \leq x$ .
2. إذا كان  $1 \leq k \leq i$  فإن  $A[k] > x$ .
3. إذا كان  $k = r$  فإن  $A[k] = x$ .



**الشكل 1.7** عملية تطبيق التحزبة PARTITION على صيغة عينة. يصبح عنصر الصيغة  $A[r]$  العنصر المحوري  $x$ . تقع جميع عناصر الصيغة الخفيفة التظليل في الجزء الأول، وقيمها لا تتجاوز  $x$ . وتقع العناصر الشديدة التظليل في الجزء الثاني، وفيها أكبر من  $x$ . لم نوضح العناصر غير المظلمة بعد في أول جزأين، والعنصر الأبيض الأخير هو المحور  $x$ . (أ) الصيغة البدائية ومواقع المتحولات. لم يوضع أي من العناصر في أول جزأين. (ب) جرى تبديل موقع القيمة 2 "مع نفسه"، ووضعت في الجزء الخاص بالقيم الصغرى. (ت) - (ث) أضيفت القيمتان 8 و 7 إلى الجزء الخاص بالقيم الكبرى. (ج) جرى تبديل موقعي القيمتين 1 و 8، وكثير جزء القيم الصغرى. (ح) جرى تبديل موقعي 3 و 7، وكثير جزء القيم الصغرى. (خ) - (د) كثير الجزء الأكبر ليتضمن 5 و 6 وننتهي الحلقة. (ذ) في السطرين 7-8، جرى تبديل موقع المحور بحيث يقع بين الجزأين.

لا تُغطّي الأدلة الواقعة بين  $f$  و  $r-1$  بأي من الحالات الثلاث، وليس لقيم هذه العناصر أية علاقة مع المحور  $x$ .

ينبغي أن نبين أن لامتغير الحلقة هنا صحيح true قبل التكرار الأول، وأن كل تكرار هذه الحلقة يحافظ على هذا الالامتغير، الذي يقدم خواص مفيدة لبيان الصحة correctness عند انتهاء الحلقة.

**الاستدعاء:** لدينا قبل التكرار الأول للحلقة،  $i = p-1$  و  $f = p$ . وحيث إنه لا توجد قيم بين  $p$  و  $i$ ، ولا قيم بين  $i+1$  و  $f-1$ ، فإن أول شرطين خاصتين بلامتغير الحلقة محققان بديهياً. يُحقق الإنساذ في السطر 1 الشرط الثالث.



**الشكل 2.7** المناطق الأربع التي يحافظ عليها إجراء PARTITION ضمن الصفيفة الجزئية  $A[p..r]$ . جميع القيم في  $A[p..i]$  أصغر من  $x$  أو تساويها، وجميع القيم في  $A[i+1..j-1]$  أكبر من  $x$ ، و  $A[j..r] = x$ . يمكن أن تأخذ العناصر في الصفيفة الجزئية  $A[j..r-1]$  أية قيمة.

**المحافظة:** يبين الشكل 3.7، أننا نعتبر حالتين اعتماداً على خرج الاختبار في السطر 4. يبين الشكل 3.7(أ) ماذا يحدث عندما يكون  $A[j] > x$ ؛ الفعل الوحيد في الحلقة هو زيادة قيمة  $j$ . بعد زيادة  $j$ ، يبقى الشرط 2 محققاً في حالة  $A[j-1]$ ، وتبقى جميع العناصر الأخرى دون تعديل. يبين الشكل 3.7(ب) ماذا يحدث عندما تكون  $A[j] \leq x$ ؛ حيث تزيد الحلقة قيمة  $i$ ، وتبدل موقعي  $A[i]$  و  $A[j]$ ، ثم تزيد قيمة  $j$ . لدينا الآن، بسبب تبديل المواقع،  $A[i] \leq x$ ، ويكون الشرط 1 محققاً. وبالمثل، يكون لدينا أيضاً  $A[j-1] > x$ ، لأن العنصر الذي وضعت لامتغير  $i$  الحلقة في  $A[j-1]$  أكبر من  $x$ .

**الانتهاء:** عند الانتهاء يكون لدينا  $j = r$ . لذا، يوجد كلاً عنصر في الصفيفة في إحدى المجموعات الثلاث الموصوفة بالامتغير، ونكون قد جزأنا القيم في الصفيفة إلى ثلاث مجموعات: أقل أو تساوي  $x$ ، وأكبر من  $x$ ، ومجموعة فيها عنصر وحيد هو  $x$ .

يبدل السطران الأخيران من PARTITION في النهاية العنصر المحوري بالعنصر الموجود في أقصى يسار المجموعة التي فيها أكبر من  $x$ ، وبذلك تنقل المحور إلى مكانه الصحيح في الصفيفة الجزئية، ثم تعيد الدليل الجديد للمحور. يحقق خرج PARTITION الآن المواصفات المحددة لمرحلة "فرق". في الحقيقة، يحقق الخرج شرطاً أقوى بقليل: بعد السطر 2 من QUICKSORT، يكون  $A[q]$  أصغر تماماً من أي عنصر في  $A[q+1..r]$ .

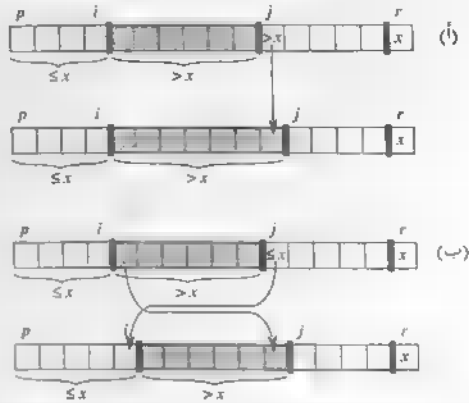
إن زمن تنفيذ PARTITION على الصفيفة الجزئية  $A[p..r]$  هو  $\Theta(n)$ ، حيث  $n = r - p + 1$  (انظر التمرين 7.1-3).

تعاريف

1-1.7

وضّح، باستخدام الشكل 1.7 نموذجاً، كيفية تطبيق التحزبة PARTITION على الصفيفة

$A = \{13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11\}$



**الشكل 3.7** حالتنا تكرار واحد للإجراء PARTITION. (أ) إذا كان  $x > A[j]$ ، فالفعل الوحيد هو زيادة قيمة  $j$  الذي يحافظ على لامتغير الحلقة. (ب) إذا كان  $x \leq A[j]$ ، تُزاد قيمة الدليل  $i$ ، ثم يجري تبديل مكاني العنصرين  $A[i]$  و  $A[j]$ ، ثم تُزاد قيمة  $j$ . جرت المحافظة على لامتغير الحلقة ثانيةً.

### 2-1.7

ما قيمة  $q$  التي يعيدها إجراء PARTITION عندما تكون قيم جميع عناصر الصفيفة  $A[p..r]$  متساوية؟ عدّل PARTITION بحيث يكون  $q = \lfloor (p+r)/2 \rfloor$  عندما تكون قيم جميع العناصر في الصفيفة  $A[p..r]$  متساوية.

### 3-1.7

أعط برهاناً مختصراً على أن زمن تنفيذ PARTITION على صفيفة جزئية طولها  $n$  هو  $\Theta(n)$ .

### 4-1.7

كيف يمكنك تعديل QUICKSORT لإجراء الفرز وفق الترتيب المتناقص؟

## أداء الفرز السريع

2.7

يعتمد زمن تنفيذ الفرز السريع على كون الشجرة متوازنة أو غير متوازنة، وهذا بدوره يعتمد على العناصر المستخدمة في الشجرة. فإذا كانت الشجرة متوازنة، تكون سرعة تنفيذ الخوارزمية مقاربة لسرعة البحث بالمرج. أما إذا كانت غير متوازنة، فيمكن أن تتفقد الخوارزمية ببطء مقارب للفرز بالإدراج. سنبحث في هذا المقطع، بصورة غير رسمية (مفصلة)، في أداء الفرز السريع بافتراض أن الشجرة المتوازنة مقارنته بأدائه عند الشجرة غير المتوازنة.

### التجزئة في أسوأ الحالات

نحدث أسوأ حالات الفرز السريع عندما يوَلَّد مسائلُ التجزئة مسألةً جزئيةً فيها  $n - 1$  عنصرًا ومسألةً ليس فيها أي عنصر. (تُلبَّث هذا الطرح في المقطع 4.7.1). لنفترض أن هذه التجزئة غير المتوازنة ستحدث في كل استدعاء عودي. تستغرق التجزئة زمانًا  $\Theta(n)$ . ولما كان تطبيق الاستدعاء العودي على صفيغة طولها 0 يخرج من الإجراء فقط، فإن  $T(0) = \Theta(1)$ ، وتكون العلاقة التكرارية لزمان التنفيذ هي:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) . \end{aligned}$$

بديهيًا، إذا جمعنا الأزمنة التي يستغرقها كلُّ مستوى من العودية، نحصل على سلسلة حسابية (المعادلة 2.1) قيمتها  $\Theta(n^2)$ . بالفعل، يمكن استخدام طريقة التعويض مباشرة لإثبات أن حل العلاقة التكرارية  $T(n) = T(n-1) + \Theta(n)$  هو  $T(n) = \Theta(n^2)$ . (انظر التمرين 2.7.1).

أي إنه، إذا كانت التجزئة غير متوازنة كليًا في كل مستوى عودي من الخوارزمية، فإن زمن التنفيذ هو  $\Theta(n^2)$ . ولذلك يكون زمن تنفيذ الفرز السريع في أسوأ الحالات ليس أفضل من الفرز بالإدراج. إضافة إلى ذلك، يكون زمن التنفيذ  $\Theta(n^2)$  عندما تكون صفيغة الدخول مفروزة كليًا سلفًا - وهي حالة شائعة زمن تنفيذ الفرز بالإدراج فيها هو  $O(n)$ .

### التجزئة في أحسن الحالات

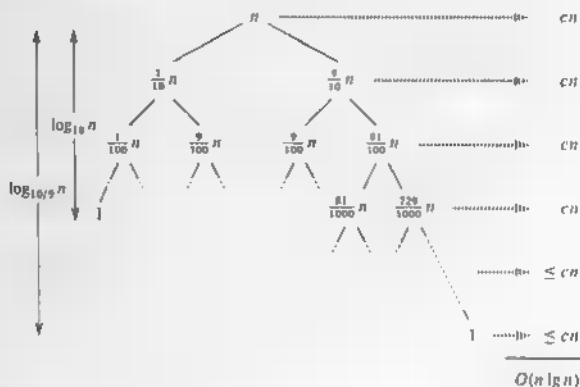
يُوَلَّد PARTITION، في الحالة التي يجري فيها التفريق إلى جزأين أكثر ما يكونان متوازنين، مسألتين فرعيتين، لا يتجاوز طول كل منهما  $n/2$ ، حيث إن طول إحداها  $\lfloor n/2 \rfloor$  وطول الثانية  $\lfloor n/2 \rfloor - 1$ . في هذه الحالة، تُنفَّذ خوارزمية الفرز السريع أسرع بكثير. وتكون للمعادلة التكرارية لزمان التنفيذ عندها:

$$T(n) = 2T(n/2) + \Theta(n) ,$$

حيث نتساهل في عدم الدقة الناتج من إهمال دالة الأرضية floor ودالة السقف ceiling ومن طرح القيمة 1. حلُّ هذه المعادلة التكرارية حسب الحالة الثانية من النظرية العامة (النظرية 4.4) هو  $T(n) = \Theta(n \lg n)$ . وهكذا، فإن موازنة طرزي التجزئة بصورة متساوية في كل مستوى من العودية تعطينا خوارزميةً أسرع تقاربيًا.

### التجزئة المتوازنة

إن زمن التنفيذ في الحالة الوسطى للفرز السريع أقرب كثيرًا إلى الحالة المثلى منه إلى أسوأ الحالات، وهو ما ستيه التحليلات في المقطع 4.7. إن مفتاح فهم السبب هو في فهم كيف يؤثر توازن التجزئة على المعادلة التكرارية التي توصف زمن التنفيذ.



**الشكل 4.7** شجرة عودية لخوارزمية QUICKSORT تولّد فيها PARTITION تفرّقاً بنسبة 1 إلى 9، وهذا يولّد زمن تنفيذ  $O(n \lg n)$ . تبيّن العقد حجّوم للسائل الجزئية، وقد جرى وضع كلفة كل مستوى إلى اليمين. تتضمن كلفة كل مستوى الثابت  $c$  الضمعي في الحد  $\Theta(n)$ .

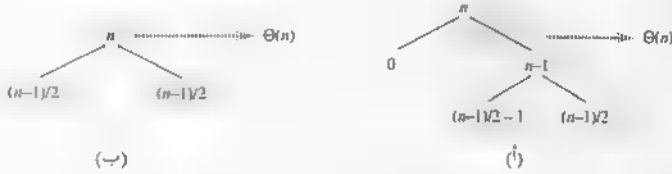
لنفترض، مثلاً، أن خوارزمية التجرئة تولّد دوّماً جزأين نسبة أحدهما إلى الآخر 9 إلى 1، الشيء الذي يبدو للوهلة الأولى غير متوازن إلى درجة كبيرة. نحصل في هذه الحالة على المعادلة التكرارية التالية لزمن تنفيذ الفرز السريع:

$$T(n) = T(9n/10) + T(n/10) + cn,$$

حيث أضفنا الثابت  $c$  صراحة بعد أن كان غفياً في الحد  $\Theta(n)$ . يبين الشكل 4.7 شجرة العودية لهذه المعادلة التكرارية. لاحظ أن كلفة كل مستوى من الشجرة تساوي  $cn$ ، إلى أن تحقّق العودية شرطاً حدّياً عند العمق  $\log_{10} n = \Theta(\lg n)$ ، تصبح بعدها كلفة كل مستوى مساوية  $cn$  على الأكثر. تنتهي العودية عند العمق  $\log_{10/9} n = \Theta(\lg n)$ . وبذلك تكون الكلفة الكلية للفرز السريع  $O(n \lg n)$ . ومن ثم، ومع تفرّق نسبته 9 إلى 1 في كل مستوى من العودية، وهو ما يبدو يدهيماً غير متوازن إلى حدّ بعيد، فإن خوارزمية الفرز السريع تُنفّذ في زمن  $O(n \lg n)$  أي في زمن مقارب للحالة التي تجري فيها عملية التفرّق في الوسط. فعلياً، حتى عملية التفرّق الذي نسبته 99 إلى 1 تتجّج زمن تنفيذ  $O(n \lg n)$ . في الحقيقة، أي تفرّق نسبته ثابتة ينتج شجرة عودية عمقها  $\Theta(\lg n)$ ، حيث كلفة كل مستوى  $O(n)$ . بالنتيجة، يكون زمن التنفيذ  $O(n \lg n)$  في حال كان التفرّق يجري بنسبة ثابتة.

### حدس بشأن الحالة الوسطى

لتكوين فكرة واضحة عن السلوك العشوائي للفرز السريع، يجب أن نضع فرضية عن مدى التواتر الذي نتوقع



**الشكل 5.7** (أ) مستويان من شجرة العودية للفرز السريع. إن تكلفة التحزبة عند الجذر هي  $n$ ، وهي تولد تقريباً "غير جيد": صيفيتان جزئيتين طولهما  $n-1$  و  $n-1$ . أما تحزبة الصفيغة الجزئية التي طولها  $n-1$ ، فتكلف  $n-1$ ، وتولد تقريباً "جيداً": صيفيتان جزئيتين طولهما  $(n-1)/2$  و  $(n-1)/2-1$ . (ب) مستوى واحد من شجرة العودية متوازن جداً. في كلا الجوانب، كلفة تحزبة المسائل الجزئية المنطلقة بشكل يضيوي مقليل هي  $\Theta(n)$ . ولكن المسائلين الجزئيتين الواحدين حلتهما في (أ) والمثلثتين بمربعين مقليلين ليسا أكبر من المسائلين الجزئيتين اللتين يلزم حلهما في (ب).

أن تصادف فيه المدخلات المختلفة. يعتمد سلوك الفرز السريع على الترتيب النسبي للقيم في عناصر الصفيغة المعطاة باعتبارها دخلاً، وليس على القيم الخاصة في الصفيغة. سنفترض، كما في تحليلنا الاحتمالي لمسألة التوظيف في المقطع 2.5، أن كل تبادل أعداد الدخيل متساوية الاحتمال.

عندما ننفذ الفرز السريع على صفيغة دخل قيمها عشوائية، فمن غير المحتمل أن تجري التحزبة بالطريقة نفسها في كل مستوى، كما افترض تحليلنا غير الرسمي. من المنطقي أن نتوقع أن بعض حالات التفريق ستكون جيدة التوازن، وبعضها ستكون متوازنة قليلاً. على سبيل المثال، يُطلب إليك في التمرين 2.7-6 إثبات أنه في 80 بالمئة من المرات تقريباً يولد PARTITION تقريباً أكثر توازناً من 9 إلى 1، وفي 20 بالمئة من المرات تقريباً يولد تقريباً أقل توازناً من 9 إلى 1.

تولد PARTITION، في الحالة الوسطى، مزيجاً من التفاريق "الجيدة" و "غير الجيدة". وفي الشجرة العودية الخاصة بتنفيذ PARTITION في الحالة الوسطى، تتوزع التفاريق الجيدة وغير الجيدة عشوائياً في جميع أرجاء الشجرة. ولكن، لنفترض للتبسيط أن التفاريق الجيدة وغير الجيدة تتبادلان المستويات في الشجرة، وأن التفاريق الجيدة هي من أحسن الحالات، و التفاريق غير الجيدة هي من أسوأ الحالات. يبين الشكل 5.7(أ) التفاريق على مستويين متتاليين من شجرة العودية. إن كلفة التحزبة عند جذر الشجرة هي  $n$ ، وطول الصفيغتين الجزئيتين هو  $n-1$  و  $0$ : وفق أسوأ الحالات. وفي للمستوى التالي، تحزب الصفيغة الجزئية التي طولها  $n-1$  وفق أحسن الحالات إلى صفيغتين جزئيتين طولهما  $(n-1)/2-1$  و  $(n-1)/2$ . لنفترض أن كلفة الشرط الحدي هي 1 للصفيغة الجزئية التي طولها 0.

يولد تركيب التفريق غير الجيد مع التفريق الجيد ثلاث صفيغات جزئية أطولها:  $(n-1)/2-1$  و  $(n-1)/2$  و  $(n-1)$  بكلفة تحزبة مركبة تساوي  $\Theta(n) + \Theta(n-1) + \Theta(n)$ . إن هذه الحالة ليست بالأكيد

أسوأ من تلك المذكورة في الشكل 5-7(ب)، أي حالة مستوى واحد من التحزبة التي تولّد صغيفتين جزئيتين طول كلٍّ منهما  $(n-1)/2$ ، وكلفة  $\Theta(n)$ . ومع ذلك، فإن هذه الحالة الأخيرة متوازنة! ومن البديهي أن تمتص كلفة التفريق الجيد  $\Theta(n)$  كلفة التفريق غير الجيد  $\Theta(n-1)$ ، ويكون التفريق الناتج جيداً. وهكذا، فإن زمن تنفيذ الفرز السريع - عند تبديل المستويات بين تفريق جيد وغير جيد - يماثل زمن التنفيذ عند إجراء التفريق الجيد فقط: أي  $O(n \lg n)$ ، ولكن مع ثابت أكبر قليلاً عظمي ضمن تدوين- $O$ . سنجري تحليلاً مفصلاً للحالة الوسطى للنسخة ذي العشوائية المضافة للفرز السريع في المقطع 2.4.7.

## تمارين

### 1-2.7

استخدم طريقة التعويض لإثبات أن حل للمعادلة التكرارية  $T(n) = T(n-1) + \Theta(n)$  هو  $T(n) = \Theta(n^2)$ ، حسبما ذكرنا في بداية المقطع 2.7.

### 2-2.7

ما هو زمن تنفيذ QUICKSORT عندما تكون لكل عناصر الصغيفة  $A$  القيمة نفسها؟

### 3-2.7

برهن أن زمن تنفيذ QUICKSORT هو  $\Theta(n^2)$  عندما تتضمن الصغيفة  $A$  عناصر متمايزة ومرتبّة وفق الترتيب التّزوي.

### 4-2.7

تسجل المصارف عادة التداولات على حساب ما وفق ترتيب زمن التداول، ولكن يرغب العديد من الناس أن يتلقوا بياناتهم المصرفية بحيث تكون الشيكات مرتبة وفق رقم الشيك. يحزّر الناس عادة الشيكات وفق ترتيب أرقام الشيكات، وتصرف الباعة هذه الشيكات عادة بعد فترة معقولة. إن مسألة تحويل الترتيب وفق زمن التداول إلى ترتيب وفق رقم الشيك هي مسألة فرز دخل مغرور تقريباً. برهن أن إجراء INSERTION-SORT قد يتفوّق على إجراء QUICKSORT في هذه المسألة.

### 5-2.7

نفترض أن نسبة التفاريق على كل مستوى من الفرز السريع هي  $1 - \alpha$  إلى  $\alpha$ ، حيث  $0 < \alpha \leq 1/2$  ثابت. برهن أن العمق الأسفري للورقة ما في شجرة العمودية هو تقريباً  $\lg n / \lg \alpha$  وأن العمق الأعظمي هو  $(\lg n / \lg(1 - \alpha))$  تقريباً. (لا تحتم بشأن تدوير العدد الصحيح.)

### \* 6-2.7

برهن أن احتمال أن يولّد PARTITION تقريباً أكثر توازناً من  $1 - \alpha$  إلى  $\alpha$  على صغيفة دخل عشوائية هو  $1 - 2\alpha$  تقريباً، وذلك مهما يكن الثابت  $0 < \alpha \leq 1/2$ .



### 3.7 نسخة للفرز السريع ذو عشوائية مضافة

عند سبر سلوك الحالة الوسطى للفرز السريع، افترضنا أن جميع تبديل أعداد الدخول ذات احتمال متساو. ولكننا لا نستطيع أن نتوقع تحقق ذلك دائماً في مسألة هندسية (انظر التمرين 4-2.7). يمكننا في بعض الأحيان - كما رأينا في المقطع 3.5 - إضافة عشوائية إلى خوارزمية بهدف الحصول على أداء جيد في الحالة الوسطى على جميع المدخلات. هذا وينظر كثيرون إلى نسخة الفرز السريع ذي العشوائية الناتجة على أنها خوارزمية الفرز الأنسب في حالة مدخلات كبيرة كفاية.

فمما في المقطع 3.5، بإضافة عشوائية إلى خوارزمتنا، وذلك بتبديل عناصر الدخول صراحة. وكان بإمكاننا إجراء ذلك للفرز السريع أيضاً، ولكن تقنية إضافة عشوائية مختلفة، تسمى *اختيار عينات عشوائية random sampling*، تعطي تحليلاً أبسط. بدلاً من استخدام  $A[r]$  محوراً على الدوام، نستخدم عنصرًا جري اختياره عشوائيًا من الصيغة الجزئية  $A[p..r]$ . نُجري ذلك بتبديل موقع العنصر  $A[r]$  بعنصر جري اختياره عشوائيًا من  $A[p..r]$ . يُضمن أخذ عينات عشوائية من المجال  $p, \dots, r$ ، أن يكون العنصر المحوري  $x = A[r]$  أيًا من عناصر الصيغة الجزئية التي عددها  $r - p + 1$ ، وذلك باحتمال متساو. ولما كان اختيار العنصر المحوري قد جرى عشوائيًا، فنحن نتوقع أن يكون تقريظ صيغة الدخول متوازنًا ووسطيًا إلى حدٍ معقول. إن التعديلات التي تُجرى على *PARTITION* و *QUICKSORT* طفيفة؛ ففي إجراء التجزئة الجديد، ما علينا سوى تمهيز تبديل المواقع قبل إجراء التجزئة الخافي:

**RANDOMIZED-PARTITION**( $A, p, r$ )

- 1  $i = \text{RANDOM}(p, r)$
- 2 **exchange**  $A[r]$  with  $A[i]$
- 3 **return** **PARTITION**( $A, p, r$ )

يستدعى الفرز السريع الجديد **RANDOMIZED-PARTITION** بدلاً من **PARTITION**:

**RANDOMIZED-QUICKSORT**( $A, p, r$ )

- 1 **if**  $p < r$
- 2      $q = \text{RANDOMIZED-PARTITION}(A, p, r)$
- 3     **RANDOMIZED-QUICKSORT**( $A, p, q - 1$ )
- 4     **RANDOMIZED-QUICKSORT**( $A, q + 1, r$ )

سنحلّل هذه الخوارزمية في المقطع التالي.

تمارين

I-3.7

لماذا نقوم بتحليل زمن التنفيذ المتوقع للخوارزمية ذات العشوائية المضافة وليس زمن التنفيذ في أسوأ الحالات؟

## 2-3.7

خلال تنفيذ RANDOMIZED-QUICKSORT، كم مرة يُستدعى مولّد الأعداد العشوائية RANDOM في أسوأ الحالات؟ وكم مرة في أفضل الحالات؟ أعطِ إجابتك باستخدام تدوين  $\Theta$ .

## 4.7 تحليل الفرز السريع

قدم المقطع 2.7 بعض الأمور البديهية المتعلقة بسلوك الفرز السريع في أسوأ الحالات، ولماذا نتوقع أن تتقدّ بسرعة. وفي هذا المقطع، نحلّل سلوك الفرز السريع تحليلاً دقيقاً. نبدأ بالتحليل في أسوأ الحالات، الذي ينطبق على QUICKSORT أو RANDOMIZED-QUICKSORT، ونختم بتحليل زمن التنفيذ المتوقع لإجراء RANDOMIZED-QUICKSORT.

## 1.4.7 التحليل في أسوأ الحالات

رأينا في المقطع 2.7 أن إجراء التفريق في أسوأ الحالات في كل مستوى من مستويات العودة في الفرز السريع يؤلّد زمن تنفيذ  $\Theta(n^2)$ ، وهو - بدبيهاً - زمن تنفيذ الخوارزمية في أسوأ الحالات. نبرهن فيما يلي هذه الفرضية المؤكّدة:

يمكننا، باستعمال طريقة التعويض (انظر المقطع 3.4)، أن نبين أن زمن تنفيذ الفرز السريع هو  $\Theta(n^2)$ . ليكن  $T(n)$  زمن أسوأ الحالات لإجراء QUICKSORT على دخل طوله  $n$ . لدينا المعادلة التكرارية:

$$T(n) = \max_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n), \quad (1.7)$$

حيث تقع قيمة التوسيط  $q$  بين 0 و  $n-1$ ، وذلك لأن إجراء PARTITION يؤلّد مسالكين جزئيتين طولهما الكلي  $n-1$ . نتوقع أن يكون  $T(n) \leq cn^2$  حيث  $c$  ثابت ما. وتعويض هذا التوقع في المعادلة (1.7)، نحصل على:

$$\begin{aligned} T(n) &\leq \max_{0 \leq q \leq n-1} (cq^2 + c(n-q-1)^2) + \Theta(n) \\ &= \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) + \Theta(n). \end{aligned}$$

يلعب التعبير  $q^2 + (n-q-1)^2$  قيمته العظمى على مجال التوسيط  $0 \leq q \leq n-1$  عند نقطتيه الطرفيتين. ولإثبات هذا الإدعاء، نلاحظ أن المشتق الثاني للتعبير بالنسبة إلى  $q$  موجب (انظر التمرين 3-4.7)، وهذه الملاحظة تعطينا الحدّ  $(n-1)^2 = (n-1)^2 \leq (q^2 + (n-q-1)^2) \leq \max_{0 \leq q \leq n-1} (q^2 + (n-q-1)^2) \leq n^2 - 2n + 1$  ومتابعة عملية وضع حد لـ  $T(n)$ ، نحصل على:

$$\begin{aligned} T(n) &\leq cn^2 - c(2n-1) + \Theta(n) \\ &\leq cn^2, \end{aligned}$$

وذلك لأن بإمكاننا اختيار الثابت  $c$  كبيراً كفاية بحيث يغطي الحد  $c(2n-1)$  على الحد  $\Theta(n)$ . وبذلك يكون  $T(n) = O(n^2)$ . وقد رأينا في المقطع 2.7 حالة خاصة يستغرق فيها الفرز السريع زمناً  $\Omega(n^2)$ ، وذلك عندما تكون التجزئة غير متوازنة. وبالمقابل، يُطلب إليك في التمرين 1-4.7 برهان أن للمعادلة التكرارية (1.7) حلاً يحقق  $T(n) = \Omega(n^2)$ . وبذلك، يكون زمن التنفيذ للفرز السريع في أسوأ الحالات هو  $\Theta(n^2)$ .

#### 2.4.7 زمن التنفيذ المتوقع

رأينا فيما سبق أن يكون الزمن المتوقع لتنفيذ خوارزمية RANDOMIZED-QUICKSORT في أسوأ الحالات هو  $O(n \lg n)$ : إذا كان التفريق الناتج عن RANDOMIZED-PARTITION في كل مستوى من العودية، يضع أية نسبة ثابتة من العناصر في جهة واحدة من التجزئة، يكون لشجرة العودية العمق  $\Theta(\lg n)$ ، كما يجري تنفيذ أعمال من رتبة  $O(n)$  في كل مستوى. حتى لو أضفنا بين هذه المستويات مستويات جديدة التفريق فيها قبل التوازن إلى حد بعيد، فسيبقى الزمن الكلي  $O(n \lg n)$ . يمكننا تحليل زمن التنفيذ المتوقع للإجراء RANDOMIZED-QUICKSORT بدقة إذا عرفنا كيف يعمل إجراء التجزئة أولاً، ثم استخدمنا هذه المعرفة لاستنتاج حد  $O(n \lg n)$  لزمن التنفيذ المتوقع. يولد هذا الحد الأعلى لزمن التنفيذ المتوقع، إضافة إلى الحد في أفضل الحالات  $\Theta(n \lg n)$ ، الذي ذكرناه في المقطع 2.7، زمن تنفيذ متوقع  $\Theta(n \lg n)$ . نفترض في أثناء ذلك أن قيم العناصر المفروزة متمايزة.

#### زمن التنفيذ ومقارنات

الاختلاف الوحيد بين إجرائي QUICKSORT و RANDOMIZED-QUICKSORT هو في كيفية اختيار العناصر المخونة؛ وهما متماثلان فيما عدا ذلك. لذا، يمكننا صياغة تحليلنا لإجراء RANDOMIZED-QUICKSORT بمناقشة إجرائي QUICKSORT و PARTITION، ولكن بافتراض أن اختيار العناصر المخونة يجري عشوائياً من الصيغة الخرجية المبررة إلى RANDOMIZED-QUICKSORT.

يهمس الزمن الذي يقضيه إجراء PARTITION على زمن تنفيذ QUICKSORT. ففي كل استدعاء لإجراء PARTITION، يختار عنصراً محورياً، ولا يُضخّن هذا العنصر في أيّ استدعاء عودي مستقبلي لإجراءي QUICKSORT و PARTITION. لذلك، يمكن أن يكون هناك  $\gg$  استدعاء على الأكثر لإجراء PARTITION خلال التنفيذ الكامل لخوارزمية الفرز السريع. يستغرق استدعاء واحد لإجراء PARTITION زمناً  $O(1)$  إضافة إلى زمن متناسب طردياً مع عدد تكرارات حلقة for الموجودة في الأسطر 6-3. يجري كل تكرار حلقة for هذه مقارنة في السطر 4، يقارن فيها المحور بعنصر آخر من الصيغة A. لهذا، إذا كان بإمكاننا عد العدد الكلي لمرات تنفيذ السطر 4، يمكننا وضع حدٍّ للزمن الكلي الذي تستغرقه حلقة for خلال التنفيذ الكامل لإجراء QUICKSORT.

### 1.7 توطئة

ليكن  $X$  عدد المقارنات التي أُجريت في السطر 4 من PARTITION خلال التنفيذ الكامل لإجراء QUICKSORT على صفيقة ذات  $n$  عنصرًا. عندها، يكون زمن تنفيذ QUICKSORT هو  $O(n + X)$ .

**البرهان** نستنتج من المناقشة السابقة أن الخوارزمية تستدعي PARTITION  $n$  مرةً على الأكثر، يُنجز في كلٍّ منها مقدارًا ثابت من الأعمال، ثم تنفذ حلقة for عددًا من المرات. ويُنفَّذ كلُّ تكرار لحلقة for السطر 4. ■

غايتنا إذن حساب  $X$ ، وهو عدد المقارنات الكلية للنفذة في جميع استدعاءات PARTITION. ولن نحاول تحليل كم هو عدد المقارنات التي أُجريت عند كلِّ استدعاء للإجراء PARTITION. بل، سنستنتج بدلاً من ذلك، حدثًا شموليًا لعدد المقارنات الكلية. ولنفعل ذلك، علينا أن نعرف متى نقارن الخوارزمية بين عنصرين من الصفيقة، ومتى لا نقارن. نسَمي، لسهولة التحليل، عناصر الصفيقة  $A$  بـ  $z_1, z_2, \dots, z_n$ ، حيث  $z_i$  هو العنصر ذو الترتيب  $i$  من حيث الصغر. ونعرِّف أيضًا المجموعة  $Z_{ij} = [z_i, z_{i+1}, \dots, z_j]$  بأنها مجموعة العناصر  $z_i$  و  $z_j$  وما بينهما.

والسؤال هو: متى نقارن الخوارزمية بين  $z_i$  و  $z_j$ ؟ للإجابة عن هذا السؤال، نلاحظ أولاً أنه تجري مقارنة كل زوج من العناصر فيما بينها مرةً واحدةً على الأكثر. لماذا؟ لأن العناصر تُقارن بالعنصر المحوري فقط، وبعد انتهاء استدعاء محدِّد لإجرائية PARTITION لا يقارن العنصر المحوري المستخدم في هذا الاستدعاء بأي عنصر آخر أبدًا.

نستخدم تحليلنا متحولات عشوائية مؤشر indicator random variables (انظر المقطع 2.5). نعرِّف:

$$X_{ij} = I\{z_i \text{ is compared to } z_j\},$$

حيث نعتبر المقارنات التي تجري في أي وقت خلال تنفيذ الخوارزمية، وليس خلال تكرار واحد أو استدعاء واحد لإجراء PARTITION فقط. ولما كانت مقارنة كل زوج تجري مرةً واحدةً على الأكثر، يمكننا بسهولة تقدير العدد الكلي للمقارنات في الخوارزمية:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}.$$

وبأخذ توقُّع الطرفين، ثم باستخدام خطية التوقع والتوطئة 1.5، نحصل على:

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{n-1} \sum_{j=i+1}^n X_{ij}\right] \\ &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n E[X_{ij}] \end{aligned}$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \Pr\{z_i \text{ is compared to } z_j\} \quad (2.7)$$

يبقى علينا حساب  $\Pr\{z_i \text{ is compared to } z_j\}$  أي احتمال مقارنة  $z_i$  بـ  $z_j$ . يفترض تحليلنا أن إجراء RANDOMIZED-PARTITION يختار كل محور عشوائيًا واستقلاليًا.

لندرس الحالات التي لا تجري فيها المقارنة بين عنصرين. لنأخذ دخالاً للفرز السريع الأعداد من 1 إلى 10 (بأي ترتيب كان)، ولنفترض أن أول عنصر محوري هو 7. إن أول استدعاء لإجراء PARTITION يفصل الأعداد إلى مجموعتين:  $\{1, 2, 3, 4, 5, 6\}$  و  $\{8, 9, 10\}$ . ثم تجري مقارنة العنصر 7 بجميع العناصر الأخرى، ولكن  $z_7$  تجري (ولن تجري) مقارنة أي عدد من المجموعة الأولى (وليكن 2 مثلاً) بأي عدد من المجموعة الثانية (وليكن 9 مثلاً).

ولما كنا نفترض أن قيم العناصر متمايزة بوجه عام، فإننا نُعرف، بعد اختيار المحور  $x$  حيث  $z_i < x < z_j$ ، أنه لا يمكن مقارنة  $z_i$  و  $z_j$  في أي وقت لاحق. من جهة أخرى، إذا اختير  $z_i$  محورًا قبل أي عنصر في  $z_{ij}$ ، فإن  $z_i$  سيقارن بكل عنصر في  $z_{ij}$ ، ما عدا ما عدا هو نفسه. وبالمثل، إذا اختير  $z_j$  محورًا قبل أي عنصر في  $z_{ij}$ ، فإن  $z_j$  سيقارن بكل عنصر في  $z_{ij}$ ، ما عدا ما عدا هو نفسه. في مثالنا، تجري مقارنة العنصرين 7 و 9، لأن 7 هو العنصر الأول الذي سيحري اختياره محورًا من  $z_{7,9}$ . وبالمقابل، لن تجري مقارنة 2 و 9 أبدًا، لأن أول محور يختير من  $z_{2,9}$  هو 7. وهكذا، تجري مقارنة  $z_i$  و  $z_j$  إذا فقط إذا كان العنصر الأول الذي سيحري اختياره محورًا من  $z_{ij}$  هو إما  $z_i$  أو  $z_j$ .

نحسب الآن احتمال وقوع هذا الحدث. إن المجموعة  $z_{ij}$  تكون بكاملها في الجزء نفسه، قبل اللحظة التي نختار فيها أحد عناصر  $z_{ij}$  ليكون محورًا. ولذا، فإن أي عنصر من  $z_{ij}$  يمكن أن يكون أول عنصر يُختار محورًا بالاحتمال نفسه. ولما كانت المجموعة  $z_{ij}$  تحوي  $i + j - 1$  عنصرًا، وكانت المخارر مختارة بعشوائية واستقلالية، فإن احتمال أن يكون عنصر ما هو العنصر الأول المختار ليكون محورًا هو  $1/(i + j - 1)$ . وبذلك، يكون لدينا<sup>1</sup>:

$$\begin{aligned} \Pr\{z_i \text{ is compared to } z_j\} &= \Pr\{z_i \text{ or } z_j \text{ is first pivot chosen from } z_{ij}\} \\ &= \Pr\{z_i \text{ is first pivot chosen from } z_{ij}\} \\ &\quad + \Pr\{z_j \text{ is first pivot chosen from } z_{ij}\} \end{aligned}$$

<sup>1</sup> أي إن احتمال أن يقارن  $z_i$  بـ  $z_j$  يساوي احتمال أن يكون  $z_i$  أو  $z_j$  هو أول محور جرى اختياره من  $z_{ij}$ . وهذا يساوي مجموع احتمال أن يكون  $z_i$  هو أول محور جرى اختياره من  $z_{ij}$  واحتمال أن يكون  $z_j$  هو أول محور جرى اختياره من  $z_{ij}$ .

$$= \frac{1}{j-i+1} + \frac{1}{j-i+1}$$

يمكن الانتقال من السطر الأول إلى السطر الثاني لأن الحدَّين يُقَعِي أحدهما الآخر mutually exclusive. وندمج المعادلتين (2.7) و (3.7)، نجد أن

$$E[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1}$$

يمكننا أن نقيّم هذا المجموع باستخدام تبديل المتحولات  $(k = j - i)$  والحد على السلاسل التوافقية harmonic series في المعادلة (أ.7):

$$\begin{aligned} E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^n \frac{2}{j-i+1} \\ &= \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k+1} \\ &< \sum_{i=1}^{n-1} \sum_{k=1}^n \frac{2}{k} \\ &= \sum_{i=1}^{n-1} O(\lg n) \\ &= O(n \lg n) . \end{aligned} \quad (4.7)$$

نستنتج من ذلك أنه عند استخدام RANDOMIZED-PARTITION، يكون زمن التنفيذ المتوقع للفرز السريع عندما تكون قيم العناصر متمايزة هو  $O(n \lg n)$ .

تعاريف

1-4.7

يُبيّن أن المعادلة التكرارية  $T(n) = \max_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \Theta(n)$  تحقق  $T(n) = \Omega(n^2)$ .

2-4.7

يبيّن أن زمن تنفيذ الفرز السريع في أحسن الحالات هو  $\Omega(n \lg n)$ .

3-4.7

يبيّن أن التعبير  $q^2 + (n-q-1)^2$  يصل إلى قيمته العظمى ضمن  $q = 0, 1, \dots, n-1$  عندما

تكون  $q = 0$  أو  $q = n - 1$ .

4-4.7

يَبَيِّنُ أن زمن التنفيذ المتوقع لإجراء RANDOMIZED-QUICKSORT هو  $\Omega(n \lg n)$ .

5-4.7

يمكن تحسين زمن تنفيذ الفرز السريع عمليًا بالاستفادة من زمن التنفيذ السريع للفرز بالإدراج حين يكون دخله مفروغًا "تقريبًا". عند استدعاء الفرز السريع على صفيقة جزئية عدد عناصرها أقل من  $k$  عنصرًا، دَعُهُ يَنتَهِ return دون فرز الصفيقة الجزئية. وبعد أن يعود استدعاء الفرز السريع ذو المستوى الأعلى، نَقْذُ الفرز بالإدراج على كامل الصفيقة لإغناء إجراءات الفرز. برهن أن خوارزمية الفرز هذه تُنَفَّذُ في زمن متوقع  $O(nk + n \lg(n/k))$ . كيف يجب انتقاء  $k$ ، من الناحيتين النظرية والعملية؟

\* 6-4.7

لنبحث في تعديل إجراء PARTITION بانتقاء عشوائي لثلاثة عناصر من الصفيقة  $A$ ، ثم بالتجزئة حول وسطها median (القيمة الوسطى للعناصر الثلاثة). قَرِّبْ احتمال الحصول على تفريق بنسبة  $\geq \alpha$  إلى  $(1 - \alpha)$  في أسوأ الحالات، وذلك بصيغة دالة بدلالة  $\alpha$  على المجال  $0 < \alpha < 1$ .

## مسائل

### 1-7 صحة تجزئة Hoare

إن نسخة PARTITION المعطاة في هذا الفصل ليست نسخة الخوارزمية الأصلية للتجزئة. وفيما يلي الخوارزمية الأصلية للتجزئة، ونُسَبِّ إلى C. A. R. Hoare:

HOARE-PARTITION( $A, p, r$ )

```

1   $x = A[p]$ 
2   $i = p - 1$ 
3   $j = r + 1$ 
4  while TRUE
5      repeat
6           $j = j - 1$ 
7          until  $A[j] \leq x$ 
8          repeat
9               $i = i + 1$ 
10         until  $A[i] \geq x$ 
11         if  $i < j$ 
12             exchange  $A[i]$  with  $A[j]$ 
13     else return  $j$ 
```

أ. اعرض ناتج تطبيق HOARE-PARTITION على الصيغة

$$A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21 \rangle$$

بحيث تظهر قيم الصيغة والقيم المساندة بعد كل تكرار للحلقة **while** في الأسطر 4-13.

يطلب إليك في الأسئلة الثلاثة التالية أن تقدم دليلاً محكماً على أن إجراء HOARE-PARTITION صحيح. بافتراض أن الصيغة الجزئية  $A[p..r]$  تتضمن عنصرين على الأقل، أثبت ما يلي:

ب. يحقق الدليلان  $i$  و  $j$  عدم إمكان الوصول إلى أي عنصر من  $A$  واقع خارج الصيغة الجزئية  $A[p..r]$ .

ت. عندما تنتهي HOARE-PARTITION، نعيد قيمة  $j$  بحيث تحقق  $p \leq j < r$ .

ث. عندما تنتهي HOARE-PARTITION يكون أي عنصر من  $A[p..j]$  أصغر أو يساوي أي عنصر من  $A[j+1..r]$ .

يفصل إجراء PARTITION في المقطع 1.7 قيمة المحور (الموجودة أصلاً في  $A[r]$ ) عن الجزأين اللذين تشكلهما. من ناحية أخرى، يضع إجراء HOARE-PARTITION قيمة المحور (الموجودة أصلاً في  $A[p]$ ) في أحد الجزأين  $A[p..j]$  و  $A[j+1..r]$  دوماً. ولما كانت  $p \leq j < r$ ، فهذا التفريق ليس بدهياً دوماً.

ج. أعد كتابة إجراء QUICKSORT بحيث يستخدم HOARE-PARTITION.

## 2-7 الفرز السريع في حالة تساوي قيم العناصر

يفترض تحليل الزمن المتوقع للفرز السريع ذي العشوائية المضافة في المقطع 2-4.7 أن قيم جميع العناصر متمايزة. ندرس في هذه المسألة ماذا يحدث عندما لا تكون كذلك.

أ. افترض أن قيم جميع العناصر متساوية. ماذا يمكن أن يكون زمن تنفيذ الفرز السريع ذي العشوائية المضافة في هذه الحالة.

ب. يعيد إجراء PARTITION دليلاً  $q$  بحيث أن قيمة كل عنصر في  $[p..q-1]$  أصغر أو تساوي  $A[q]$  وقيمة كل عنصر في  $A[q+1..r]$  أكبر من  $A[q]$ . عدّل إجراء PARTITION لإنشاء إجراء  $\text{PARTITION}'(A, p, r)$ ، الذي يبدل مواقع عناصر  $A[p..r]$  ويعيد دليلين  $q$  و  $t$ ، حيث  $p \leq q \leq t \leq r$ ، وبحيث تكون

■ جميع عناصر  $A[q..t]$  متساوية،

• قيمة كل عنصر في  $[p..q-1]$  أصغر من  $A[q]$ ،

• وقيمة كل عنصر في  $A[t+1..r]$  أكبر من  $A[q]$ .



وكما في PARTITION يجب أن يستغرق تنفيذ إجراء 'PARTITION' زمناً  $\Theta(r - p)$ .

ت. عدّل إجراء RANDOMIZED-PARTITION بحيث يستدعي 'PARTITION'، وسمِّ الإجراء الجديد 'RANDOMIZED-PARTITION'. ثم عدّل إجراء QUICKSORT لإنشاء إجراء 'QUICKSORT'(A, p, r) يستدعي 'RANDOMIZED-PARTITION' وينطبق عودياً فقط على الأجزاء التي لا تُعرف عناصرها بأنها متساوية فيما بينها.

ث. باستخدام 'QUICKSORT'، كيف يمكنك مواءمة التحليل في المقطع 2-4.7 لتُحسب افتراض أن كل العناصر متمايزة؟

### 3-7 تحليل بديل للفرز السريع

يركز تحليل بديل لزمّن تنفيذ الفرز السريع ذي العشوائية للضافة على زمن التنفيذ المتوقع لكل استدعاء عودي مستقل لإجراء RANDOMIZED-QUICKSORT، بدلاً من عدد المقارنات المنفذة.

أ. ناقش أنه إذا كانت لدينا صيغة طولها  $n$ ، فإن احتمال أن يجري اختيار عنصر ما محوراً هو  $1/n$ . استخدم هذا لتعريف متحولات عشوائية مؤشرة  $X_i = I$  (يجري اختيار العنصر ذي الترتيب  $i$  من حيث الصغر محوراً). ما هو  $E[X_i]$ ؟

ب. ليكن  $T(n)$  متحولاً عشوائياً يرمز إلى زمن تنفيذ الفرز السريع على صيغة طولها  $n$ . ناقش أن:

$$E[T(n)] = E\left[\sum_{q=1}^n X_q(T(q-1) + T(n-q) + \Theta(n))\right]. \quad (5.7)$$

ت. بيّن أن بالإمكان كتابة المعادلة (5.7) على الشكل:

$$E[T(n)] = \frac{2}{n} \sum_{q=2}^{n-1} E[T(q)] + \Theta(n) \quad (6.7)$$

ث. بيّن أن:

$$\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \quad (7.7)$$

(تلميح: افصل المجموع إلى جزأين، الأول لعيم  $k = 2, 3, \dots, [n/2] - 1$  والآخر لعيم  $k = [n/2], \dots, n-1$ ).

ج. بيّن باستخدام الحد الموجود في المعادلة (7.7) أن للمعادلة التكرارية (6.7) حلاً  $E[T(n)] = \Theta(n \lg n)$ . (تلميح: أثبت بالتعويض، أن  $E[T(n)] \leq an \lg n$  في حالة كون  $n$  كبيرة كفاية و  $a$  ثابتاً موجباً).

## 4-7 عمق المكس في الفرز السريع

تتضمن خوارزمية QUICKSORT في القطع 1.7 استدعاءً عوديين لنفسها. بعد استدعاء QUICKSORT لإجراء PARTITION يفرز عودياً الصفيقة الجزئية اليسرى ثم الصفيقة الجزئية اليمنى. يعتبر الاستدعاء العودي الثاني لخوارزمية QUICKSORT غير ضروري حقاً؛ إذ يمكن تجنبه باستخدام بنية تحكم تكرارية. تتوفر هذه التقنية، التي تسمى **عودية الذيل tail recursion**، ألياً في المترجمات compilers الجيدة. لنأخذ النسخة التالية من الفرز السريع، الذي يحاكي عودية الذيل.

```

TAIL-RECURSIVE-QUICKSORT( $A, p, r$ )
1  while  $p < r$ 
2      // Partition and sort left subarray
3       $q = \text{PARTITION}(A, p, r)$ 
4      TAIL-RECURSIVE-QUICKSORT( $A, p, q - 1$ )
5       $p = q + 1$ 

```

أ. نأش أن  $\text{TAIL-RECURSIVE-QUICKSORT}(A, 1, A.length)$  تفرز الصفيقة  $A$  فرزاً صحيحاً.

تتخذ المترجمات عادةً الإجراءات العودية باستخدام مكس **Stack** يحوي معلومات متعلقة بكل استدعاء عودي، ومن ضمنها قيم المُوسِّطات parameter values. توجد للمعلومات المتعلقة بأحدث استدعاء على قمة المكس، والمعلومات المتعلقة بالاستدعاء الابتدائي في الأسفل. عند طلب الإجراء، يجري دفع معلوماته في المكس؛ وعندما تنتهي، تُنزع معلوماته من المكس. ولما كنا نفترض أن موسِّطات الصفيقة تُنقل بمؤشرات، فإن المعلومات المتعلقة بكل استدعاء إجراء على المكس تتطلب حجم تخزين  $O(1)$  في المكس. يُعرَّف **عمق المكس stack depth** بأنه مقدار حجم التخزين الأعظمي المستخدم في أي وقت أثناء الحساب.

ب. جيف مشهداً يكون فيه عمق المكس في  $\text{TAIL-RECURSIVE-QUICKSORT}$   $\Theta(n)$  على صفيقة دحل ذات  $n$  عنصراً.

ت. عدّل رماز  $\text{TAIL-RECURSIVE-QUICKSORT}$  بحيث يكون عمق المكس في أسوأ الحالات  $\Theta(\lg n)$ . حافظ على زمن تنفيذ متوقع للخوارزمية  $O(n \lg n)$ .

## 5-7 التجزئة وفق وسط ثلاثة عناصر

إحدى طرائق تحسين إجراء  $\text{RANDOMIZED-QUICKSORT}$  هي إجراء التجزئة حول محور يجري اختياره بعناية أكبر من مجرد أخذ عنصر عشوائياً من الصفيقة الجزئية. إحدى للنهيجات الشائعة هي طريقة **وسط الثلاثة median-of-3**: اختر المحور هو العنصر الوسط median (العنصر الأوسط middle) من مجموعة مولقة من 3 عناصر جرى اختيارها عشوائياً من الصفيقة الجزئية. (انظر التمرين 4.7-6). لنفترض، في هذه للسألة، أن

عناصر صفيقة الدخل  $A[1..n]$  متمايزة وأن  $n \geq 3$ . نرسم لصفيقة المخرج المفروزة بالرمز  $A'[1..n]$  عرّف، باستخدام طريقة وسط الثلاثة لاختيار المحور  $x$ ، القيمة  $p_i = \Pr(x = A'[i])$ .

أ. أعط الصفيقة الدقيقة لقيم  $p_i$  كدالة في  $n$  و  $i$  لقيم  $i = 2, 3, \dots, n-1$ . (لاحظ أن  $p_1 = p_n = 0$ ).

ب. بكم زدنا إمكان likelihood كون المحور  $x = A'[(n+1)/2]$  هو وسط الصفيقة  $A[1..n]$  مقارنة بالتنجيز العادي؟ افترض أن  $n \rightarrow \infty$ ، وأعط نسبة حدية لهذا الاحتمال.

ت. إذا عرفنا التفريق "الجيد" بأنه اختيار المحور  $x = A'[i]$  حيث  $n/3 \leq i \leq 2n/3$ ، بكم نكون قد زدنا إمكان الحصول على تفريق جيد مقارنة بالتنجيز العادي؟ (لمصيح: قُرب المجموع إلى تكامل).

ث. ناقش كون طريقة وسط الثلاثة تؤثر فقط على العامل الثابت في زمن تنفيذ الفرز السريع  $\Omega(n \lg n)$ .

### 6-7 الفرز الترجيحي للمجالات

لنأخذ مسألة فرز لا تُعرف فيها الأعداد بالتحديد. بل نعرف أن لكل عدد مجالاً ينتمي إليه على مستقيم الأعداد الحقيقية. أي لدينا  $n$  مجالاً مغلقاً من الشكل  $[a_i, b_i]$ ، حيث  $a_i \leq b_i$ . المهدف هنا هو فرز هذه المجالات فرزاً ترجيحياً *fuzzy-sort*، أي إيجاد تبديل permutation  $\langle i_1, i_2, \dots, i_n \rangle$  من المجالات بحيث أنه في حالة  $z = 1, 2, \dots, n$  يوجد  $c_z \in [a_{i_z}, b_{i_z}]$  يحقق  $c_1 \leq c_2 \leq \dots \leq c_n$ .

أ. صمّم خوارزمية ذات عشوائية مضافة لفرز  $n$  مجالاً ترجيحياً. يجب أن يكون لخوارزمتك البنية العامة لخوارزمية تفرز سريعاً النقاط المحدبة اليسرى (قيم  $a_i$ )، ولكنها يجب أن تستفيد من المجالات المتراكبة overlapping لتحسين زمن التنفيذ. (كلما تراكبت المجالات أكثر، أصبحت مسألة فرز المجالات ترجيحياً أسهل. يجب أن تستفيد خوارزمتك من هذا التراكب إلى أقصى حد).

ب. برهن أن خوارزمتك هذه تُنفَّذ بزمن متوقع  $\Theta(n \lg n)$  في الحالة العامة، ولكنها تُنفَّذ في زمن متوقع  $\Theta(n)$  عندما تراكب جميع المجالات (أي عندما توجد قيمة  $x$  بحيث يكون  $x \in [a_i, b_i]$  لكل قيم  $i$ ). يجب ألا نتحقق خوارزمتك من هذه الحالة بصورة صريحة، بل يجب أن يتحسن أدائها طبيعياً عندما يزداد حجم التراكب.

## ملاحظات الفصل

ابتكر Hoare [170] إجراء الفرز السريع؛ وقد عرضنا نسخته في المسألة 1.7. ويعود الفضل في إجراء PARTITION المذكور في المقطع 1.7 إلى N. Lomuto. ويعود التحليل في المقطع 4.7 إلى Avrim Blum.

يقدم Sedgewick [305] و Bentley [43] مرجعاً جيداً حول تفاصيل التنجيز ومدى أهميتها. بين McIlroy [248] كيفية هندسة "خصم قاتل killer adversary" يولد صفيقةً يستغرق أي تنجيز للفرز السريع عليها، افتراضياً، زمناً  $\Theta(n^2)$ . إذا كان التنجيز ذا عشوائية مضافة، فإن الخصم يولد الصفيقة بعد مشاهدة الخيارات العشوائية من خوارزمية الفرز السريع.

عرضنا حتى الآن العديد من الخوارزميات التي تستطيع فرز  $n$  عددًا في زمن  $O(n \lg n)$ . يُجرى الفرز بالدمج والفرز بالكومة هذا الحد الأعلى في أسوأ الحالات؛ في حين يُجرى الفرز السريع هذا الحد على نحو وسطي. إضافةً إلى ذلك، يمكننا، في كل خوارزمية من هذه الخوارزميات، إيجاد متتالية من  $n$  عددًا تؤدي إلى تنفيذ الخوارزمية في زمن  $\Omega(n \lg n)$ .

تشارك هذه الخوارزميات بخاصية جديدة بالاهتمام: يعتمد الترتيب المفروض الذي تحدده هذه الخوارزميات فقط على المقارنات بين عناصر الدخل. نُسَمَّى مثل خوارزميات الفرز هذه بـ **الفرز بالمقارنة** *comparison sorts*. إن جميع خوارزميات الفرز المقدمة حتى الآن هي من نوع الفرز بالمقارنة.

نبرهن في المقطع 1.8 أن أية خوارزمية فرز بالمقارنة يجب أن تُجري  $\Omega(n \lg n)$  عملية مقارنة في أسوأ الحالات لفرز  $n$  عنصرًا. وعلى ذلك، فإن الفرز بالدمج والفرز بالكومة أمثلتَيْن بالمقارنة، ولا توجد خوارزميات فرز بالمقارنة أسرع منهما بأكثر من معامل ثابت.

تناقش المقاطع 2.8 و 3.8 و 4.8 ثلاث خوارزميات فرز تُنفَّذ في زمن خطي؛ وهي: الفرز بالعدّ counting sort، والفرز حسب الأساس radix sort، والفرز بالدلاء bucket sort. نستخدم هذه الخوارزميات، طبقًا، عمليات غير عمليات المقارنة لتحديد الترتيب المفروض. لذلك، فإن الحد الأدنى  $\Omega(n \lg n)$  لا ينطبق عليها.

## 1.8 الحدود الدنيا للفرز

لا يُجرى في الفرز بالمقارنة سوى مقارنات بين العناصر، وذلك للحصول على معلومات عن ترتيب متتالية دخل  $\{a_1, a_2, \dots, a_n\}$ . أي، إذا كان لدينا العنصران  $a_i$  و  $a_j$ ، فإننا نحز أحد الاختبارات:  $a_i < a_j$  أو  $a_i \leq a_j$  أو  $a_i = a_j$  أو  $a_i \geq a_j$  أو  $a_i > a_j$  لتحديد ترتيبها النسبي. ولا يمكننا فحص قيم العناصر أو الحصول على معلومات عن ترتيبها بأية طريقة أخرى.

سنفترض في هذا المقطع، دون أن يؤثر ذلك على العمومية، أن جميع عناصر الدخل متميزة. إذا أخذنا

هذه الفرضية بالحسبان، فإن مقارنات من الشكل  $a_i = a_j$  ستكون عديدة الجدوى، لذا يمكننا أن نفترض أنه لن نجرى مقارنات من هذا الشكل. نُشير أيضًا إلى أن المقارنات  $a_i \leq a_j$  و  $a_i \geq a_j$  و  $a_i > a_j$  و  $a_i < a_j$  جميعها متكافئة لكونها تقدم معلومات متطابقة عن الترتيب النسبي لـ  $a_i$  و  $a_j$ . لذلك، فإننا سنفترض أن جميع المقارنات هي من الشكل  $a_i \leq a_j$ .

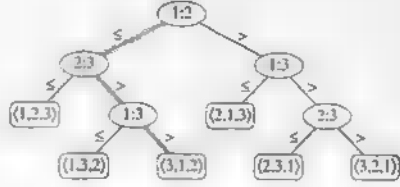
### نموذج شجرة القرار

يمكننا دراسة الفرز بالمقارنة على نحو مجرد باستخدام أشجار القرار. شجرة القرار *decision trees* هي شجرة ثنائية مملأة تمثل المقارنات بين العناصر التي تُستخدَم باستخدام خوارزمية فرز مُحددة تعمل على دَخلٍ حَجمه معطى. يتجاهل هذا التمثيل عمليات التحكم وتحريك للمعطيات وجميع الجوانب الأخرى للخوارزمية. يبين الشكل 1.8 شجرة قرار مقابلة لخوارزمية الفرز بالإدراج، للشروحة في المقطع 1.2، تعمل على متتالية دخل من ثلاثة عناصر.

نُشير، في شجرة القرار، إلى كل عقدة داخلية بـ  $z$ : لقيمة ما لـ  $t$  و  $z$  ضمن المجال  $1 \leq i, j \leq n$ ، حيث  $n$  عدد العناصر في متتالية الدخل. ونُشير أيضًا إلى كل ورقة بتبديل  $\{\pi(1), \pi(2), \dots, \pi(n)\}$ . (انظر المقطع 1.1 للاطلاع على التباديل.) يقابل تنفيذ خوارزمية الفرز تعقُّب مسار بسيط ما من جذر شجرة القرار نزولاً إلى أحد الأوراق. تُشير كل عقدة داخلية إلى المقارنة  $a_i \leq a_j$ . تُعَلِي بعدها الشجرة الجزئية اليسرى مقارنات تالية عندما  $a_i \leq a_j$ ، في حين تُعَلِي الشجرة الجزئية اليمنى مقارنات تالية عندما  $a_i > a_j$ . عندما نصل إلى ورقة ما، تكون خوارزمية الفرز قد أُرست الترتيب  $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$ . وحيث إنه يجب أن تكون أية خوارزمية فرز صحيحة قادرة على إنتاج كل تبديل من تبديل دخلها، فإن كل تبديل من تبديل  $n$  عنصرًا (وعددتها  $n!$ ) يجب أن يظهر على أنه إحدى أوراق شجرة القرار حتى يكون الفرز بالمقارنة صحيحًا. إضافة إلى ذلك، يجب أن يكون من الممكن الوصول إلى كل من هذه الأوراق، بدءًا من الجذر عبر مسار نازلٍ مقابلٍ لتنفيذ فعليٍّ للفرز بالمقارنة. (تُصِفُ مثل هذه الأوراق بأعما "أوراق يمكن الوصول إليها *reachable*".) لذا، فإننا سنفسر دراستنا على أشجار القرار التي يظهر فيها كل تبديل على أنه ورقة يمكن الوصول إليها.

### حد أدنى لأسوأ الحالات

إن طول أطول مسار بسيط من جذر شجرة قرار إلى أيٍّ من أوراقها التي يمكن الوصول إليها يُعَدُّ عدد المقارنات التي تنفذها خوارزمية البحث للمقابلة في أسوأ الحالات. لذلك، فإن عدد المقارنات في أسوأ الحالات لخوارزمية فرز بالمقارنة يساوي ارتفاع شجرة قرارها. إن حدًا أدنى لارتفاعات جميع أشجار القرار التي يظهر فيها كل تبديل على أنه ورقة يمكن الوصول إليها هو إذن حدٌ أدنى لزمَن تنفيذ أية خوارزمية بحثٍ بالمقارنة. تُعطي المرحلة التالية مثل هذا الحد الأدنى.



**الشكل 1.8** شجرة القرار لفرز بالإدراج يعمل على ثلاثة عناصر. تشير العقدة الداخلية المشار إليها بـ  $i, j$  إلى مقارنة بين  $a_i$  و  $a_j$ . وتشير الورقة المشار إليها بالتبديل  $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$  إلى الترتيب المقارن بين  $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$ . ويشير المسار المظلل إلى القرارات المتخذة عند فرز متتالية الدخول  $\langle a_1 = 6, a_2 = 8, a_3 = 5 \rangle$  ويشير التبديل  $\langle 3, 1, 2 \rangle$  عند الورقة إلى أن الترتيب المفروض هو  $a_3 = 5 \leq a_1 = 6 \leq a_2 = 8$ . لدينا  $3! = 6$  تبديلات ممكنة لعناصر الدخول، لذلك يجب أن تحتوي شجرة القرار على الأقل على 6 أوراق.

### مبرهنة 1.8

نحتاج أية خوارزمية بحث بالمقارنة إلى  $\Omega(n \lg n)$  عملية مقارنة في أسوأ الحالات.

**البرهان** يكفي، من المناقشة السابقة، تحديد ارتفاع شجرة القرار التي تظهر فيها كل تبديل على أنه ورقة يمكن الوصول إليها. لنكن لدينا شجرة قرار ارتفاعها  $h$  ولها  $l$  ورقة يمكن الوصول إليها، وهي شجرة تقابل فرزًا بالمقارنة على  $n$  عنصرًا. ولما كان كل تبديل من تبديلات الدخول (وعددتها  $n!$ ) يظهر على أنه إحدى الأوراق، فإن  $l \leq n!$ . ولما كانت أية شجرة ثنائية ارتفاعها  $h$ ، لا تزيد عدد أوراقها على  $2^h$  ورقة، فإن لدينا

$$n! \leq l \leq 2^h,$$

وبأخذ اللوغاريتم الطرفين نجد:

$$h > \lg(n!) \quad (\text{لأن الدالة } \lg \text{ متزايدة بانتظام})$$

$$> \Omega(n \lg n) \quad ((19.3) \text{ من المعادلة})$$

### نتيجة 2.8

الفرز بالكومة والفرز بالدمج هما خوارزميتا فرز بالمقارنة أمثلتان بالمقارنة.

**البرهان** إن الحدود العليا  $O(n \lg n)$  لأزمة تنفيذ الفرز بالكومة والفرز بالدمج تطابق الحد الأدنى في أسوأ الحالات  $\Omega(n \lg n)$  من المبرهنة 1.8.

## تمارين

## 1-1.8

ما هو أصغر عمق ممكن لورقة في شجرة قرار الفرز بالمقارنة؟

## 2-1.8

احصل على حد ملاصق بالمقاربة لـ  $\lg(n!)$  دون استخدام تقريب ستيرلينغ. وذلك بأن تُقدِّر المجموع  $\sum_{k=1}^n \lg k$  باستخدام تقنيات من المقطع 2.

## 3-1.8

بيِّن أنه لا توجد خوارزمية فرز بالمقارنة زمن تنفيذها خطي لنصف المُدخلات (التي عددها  $n!$  وطولها  $n$ ) على الأقل. ماذا عن الجزء  $1/n$  من المُدخلات التي طولها  $n$ ? وماذا عن الجزء  $1/2^n$ ؟

## 4-1.8

ليكن المطلوب فرز متالية من  $n$  عنصراً. تتكون متالية الدخول من  $n/k$  متالية جزئية، تحتوي كلٌّ منها على  $k$  عنصراً. إن جميع العناصر في متالية جزئية ما أصغر من العناصر في المتالية الجزئية التالية وأكبر من العناصر في المتالية الجزئية السابقة. ومن ثم، فإن كل ما نحتاج إليه لفرز كامل المتالية بطول  $n$  هو فرز  $k$  عنصراً لجميع الـ  $n/k$  متالية جزئية. أعطِ حدًّا أدنى من الرتبة  $\Omega(n \lg k)$  لعدد المقارنات اللازمة لحلِّ هذا الصيغة المعدلة من مسألة الفرز. (نمسيح: إن جميع الحدود الدنيا لكل من المتاليات الجزئية تنقصه الدقة الرياضية.)

## الفرز بالعدد

## 2.8

يفترض **الفرز بالعدد** *counting sort* أن كلَّ عنصرٍ من عناصر الدخول ( $n$  عنصراً) هو عدّد صحيح من المجال من  $1$  إلى  $k$ ، حيث  $k$  عدد صحيح. فإذا كان  $k = O(n)$ ، فإن الفرز يُنفَّذ في زمن  $O(n)$ .

يُحدِّد الفرز بالعدد، لكلَّ عنصرٍ دخلي  $x$ ، عدّد العناصر التي هي أصغر من  $x$ . يُستخدم الفرز هذه المعلومات لوضع العنصر  $x$  مباشرةً في موقعه في صيغة المخرج. فمثلاً، إذا وُجد 17 عنصراً أقل من  $x$ ، فإن  $x$  تظهر في المخرج في الموقع 18. وعلينا تعديل هذه الآلية قليلاً لمعالجة الحالة التي يوجد فيها عدة عناصر لها القيمة نفسها، لأننا لا نريد وضعها جميعاً في الموقع نفسه.

نفترض، في رماز الفرز بالعدد، أن الدخْل هو الصيغة  $A[1..n]$ ، ومن ثم فإن  $A.length = n$ . نحتاج إلى صيفيتين إضافيتين: الصيغة  $B[1..n]$  لتخزين المخرج للفرز، والصيغة  $C[0..k]$  التي تتيح ذاكرة تخزين مؤقتة.

COUNTING-SORT( $A, B, k$ )

- 1 let  $C[0..k]$  be a new array
- 2 for  $i = 0$  to  $k$

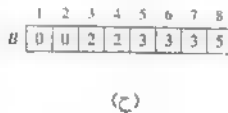
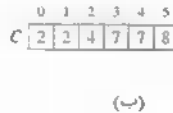
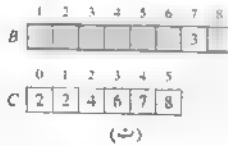


```

3      C[i] = 0
4  for j = 1 to A.length
5      C[A[j]] = C[A[j]] + 1
6  // C[i] now contains the number of elements equal to i.
7  for i = 1 to k
8      C[i] = C[i] + C[i - 1]
9  // C[i] now contains the number of elements less than or equal to i.
10 for j = A.length downto 1
11     B[C[A[j]]] = A[j]
12     C[A[j]] = C[A[j]] - 1
    
```

يوضح الشكل 2.8 الفرز بالعد. بعد حلقة **for** في السطرين 2-3 التي تسند أصنافاً إلى جميع عناصر الصيغة  $C$ ، تنحصر حلقة **for** في السطرين 4-5 قيمة كل عنصر داخل. إذا كانت قيمة عنصر دخل تساوي  $i$ ، فإننا نزيد واحدًا على  $C[i]$ . وهكذا، نحفظ  $C[i]$ ، بعد السطر 5، بعدد عناصر الدخل التي تساوي  $i$  لكل عدد صحيح  $i = 0, 1, \dots, k$ . يُحدّد السطران 6-7، لكل  $i = 0, 1, \dots, k$ ، عدد عناصر الدخل التي هي أقل من (أو تساوي)  $i$  وذلك بالقيام بجمع تراكمي **running sum** للصيغة  $C$ .

في النهاية، نضع حلقة **for** في الأسطر 10-12 كل عنصر  $A[j]$  في مكان فرزه الصحيح ضمن الصيغة  $B$ . إذا كانت جميع العناصر (وعددتها  $n$ ) متمايزة، فإننا عندما ندخل السطر 10 أول مرة، لكل عنصر  $A[j]$ ، فإن قيمة  $C[A[j]]$  تغطي المكان النهائي الصحيح لـ  $A[j]$ ، لأنه يوجد  $C[A[j]]$  عنصرًا أصغر من (أو تساوي)  $A[j]$ . ولكن لما كان من الممكن ألا تكون العناصر متمايزة، فإننا نقيس  $C[A[j]]$  في كل مرة نضع



**الشكل 2.8** تطبيق **COUNTING-SORT** على صيغة الدخل  $A[1..8]$ ، حيث كل عنصر من  $A$  هو عدد صحيح غير سالب لا يزيد على  $k = 5$ . (أ) الصيغة  $A$  والصيغة للمساعدة  $C$  بعد السطر 5. (ب) الصيغة  $C$  بعد السطر 8. (ج) صيغة المخرج  $B$  والصيغة للمساعدة  $C$  بعد تكرار الحلقة في الأسطر 10-12 مرة ومرتين وثلاث مرات، على الترتيب. العناصر للظلال قليلًا فقط من الصيغة  $B$  هي التي جرى تعبئتها. (د) صيغة المخرج النهائية للترتيب  $B$ .

القيمة  $A[j]$  في الصيغة  $B$ . يؤدي إنقاص  $C[A[j]]$  إلى وضع عنصر تالي له قيمة  $A[j]$  نفسها - إن وُجد هذا العنصر - قبل موقع  $A[j]$  تمامًا في صيغة المخرج.

كم من الوقت يتطلب تنفيذ الفرز بالعدد؟ تستغرق حلقة **for** في السطرين 2-3 زمنًا  $\Theta(k)$ ، وفي السطرين 4-5 زمنًا  $\Theta(n)$ ، وفي السطرين 7-8 زمنًا  $\Theta(k)$ ، وفي الأسطر 10-12 زمن  $\Theta(n)$ . وبذلك، يكون الزمن الكلي  $\Theta(k + n)$ . عمليًا، نستخدم الفرز بالعدد عادةً عندما يكون لدينا  $k = O(n)$ ، ويساوي زمن التنفيذ في هذه الحالة  $\Theta(n)$ .

يتغلب الفرز بالعدد على الحد الأدنى  $\Omega(n \lg n)$  للزمن في المقطع 1.8 لكونه ليس فرزًا بالمقارنة. في الواقع، ليست هناك أية مقارنة بين عناصر الدخّل في أي مكان من الرماز. عوضًا عن ذلك، يُستخدم الفرز بالعدد القيم الفعلية للعناصر كمؤشرات داخل صيغة. إن الحد الأدنى  $\Omega(n \lg n)$  للفرز لا ينطبق عندما نبتعد عن نموذج الفرز بالمقارنة.

إحدى الخواص الهامة للفرز بالعدد هي **الاستقرار stable**: حيث تظهر الأعداد التي لها القيمة نفسها في صيغة المخرج بالترتيب نفسه التي تكون عليه في صيغة الدخّل. وهذا يعني، أنه عند تساوي عددين نُعمد القاعدة التي تنصّ على أن العدد الذي يظهر أولاً في صيغة الدخّل يظهر أولاً في صيغة المخرج. هذا وتتحلّى أهميةً خاصة الاستقرار، عادةً، عندما يكون هناك معطيات تابعة للعنصر الذي يجري فرزها. ولما سبب آخر، يتعلّق بأهمية استقرار الفرز بالعدد، وهو أن الفرز بالعدد كثيرًا ما يُستخدم باعتباره مساقًا فرعيًا في الفرز حسب الأساس **radix sort**. وسنرى في المقطع التالي، أن الفرز بالعدد يجب أن يكون مستقرًا كي يكون عمل الفرز حسب الأساس صحيحًا.

تمارين

1-2.8

اشرح، بالاستعانة بالشكل 2.8 نموذجًا، غَلّ **COUNTING-SORT** على الصيغة

$$A = \langle 6, 0, 2, 0, 1, 3, 4, 6, 1, 3, 2 \rangle$$

2-2.8

برهن أن **COUNTING-SORT** مستقر.

3-2.8

افترض أننا أعدنا كتابة حلقة **for** التي تبدأ في السطر 10 لإجرائية **COUNTING-SORT** كما يلي:

10 **for**  $j = 1$  **to**  $A.length$

بيّن أن الخوارزمية سوف تبقى تعمل كما يجب. هل الخوارزمية المُعدّلة مستقرة؟

4-2.8

صِف، لكل عدد صحيح معطى  $n$  ضمن المجال 0 إلى  $k$ ، خوارزمية تعالج دُخْلها سبقيًا، ثم تجيب عن أي

استفسار عن عدد الأعداد الصحيحة من الأعداد  $n$  التي تقع ضمن المجال  $[a..b]$  في زمن  $O(1)$ . يجب أن تستغرق المعالجة السبقية في خوارزمتك زمناً  $\Theta(n + k)$ .

### 3.8 الفرز حسب الأساس

خوارزمية الفرز حسب الأساس *radix sort* هي الخوارزمية المستخدمة في آلات فرز البطاقات التي لا نجدها الآن إلا في متاحف الحواسيب. تحتوي كل بطاقة 80 عموداً، وفي كل عمود، يمكن لآلة تنقيب إحداث تقب في موقع واحد من 12 موقعاً. يمكن "برجة" الفازرة ميكانيكياً بحيث تفحص عموداً محدداً في كل بطاقة من رزمة بطاقات، ثم تضع كل بطاقة في حاوية من 12 حاوية تبعاً لموقع التقب. بعد ذلك، يُجمع عامل بطاقات الحاويات حاوية تلو الأخرى، بحيث تكون البطاقات المثقبة في الموقع الأول في أعلى البطاقات، تليها تلك المثقبة في الموقع الثاني، وهكذا دواليك.

في الأرقام العشرية، يُستخدم كل عمود 10 مواقع فقط. (نُستخدم المواقع الأخرى لترميز محارف غير رقمية). وبذلك فإن عدداً مؤلفاً من  $d$  حانة يُخزّن حقلاً مؤلفاً من  $d$  عموداً. ولما كان بمقدور فازرة البطاقات النظر إلى عمود واحد فقط في كل مرة، فإن مسألة فرز  $n$  بطاقة تمثل أعداداً من  $d$  حانة تحتاج إلى خوارزمية فرز.

من البديهي أنه يمكنك فرز الأعداد تبعاً للحنة الأعلى قيمة *most significant digit*، ثم تفرز كل حاوية من الحاويات الناتجة عودتها، وبعد ذلك تُجمع الرزم بالترتيب. ولما كان يجب وضع البطاقات في تسع حاويات من الحاويات العشر جانباً لفرز كل حاوية على حدة، فإن هذه الإجراءية، لسوء الحظ، تُؤد العديّة من كدسات البطاقات الوسيطة التي يجب عليك متابعتها. (انظر التمرين 3.8-5).

يحل الفرز حسب الأساس مسألة فرز البطاقات بطريقة غير حدية، وذلك بالفرز أولاً تبعاً للحنة الأصغر قيمة. ثم تُجمع الخوارزمية البطاقات في رزمة واحدة، بحيث تسبق البطاقات في الحاوية 0 البطاقات في الحاوية 1، والتي بدورها تسبق البطاقات في الحاوية 2، وهكذا ... ثم تُعيد فرز كامل الرزمة مرةً أخرى بحسب الحانة ذات المرتبة الثانية وتُعيد تجميع الحاويات بطريقة مشابهة. تستمر العملية حتى يجري فرز البطاقات تبعاً للحنات  $d$  كلها. من المدهش، أنه في هذه المرحلة تكون البطاقات مرتبة كلياً بحسب الأعداد من  $d$  حانة. إذن، يحتاج الفرز فقط إلى  $d$  مروراً على الرزمة. يبين الشكل 3.8 كيف يعمل الفرز حسب الأساس على "رزمة" من 7 أعداد كل منها مؤلف من 3 خانات.

وحتى يكون عمل الفرز حسب الأساس صحيحاً، يجب أن يكون فرز الحانات مستقرّاً. إن الفرز الذي تقوم به فازرة البطاقات مستقر، ولكن يجب أن يكون العامل حذراً حتى لا يغير ترتيب البطاقات عندما تخرجها من الحاوية، حتى لو كانت جميع الأوراق في الحاوية الواحدة لها الرقم نفسه في العمود المختار.

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

**الشكل 3.8** عملية الفرز حسب الأساس على لائحة مؤلفة من 7 أعداد، وكل عدد يتكون من ثلاث خانوات. يبين العمود الأيسر لائحة الدخول. في حين تبين الأعمدة اللاحقة اللوائح بعد الفرز المتتالي وفق الخانات ذات المراتب المتزايدة. يشير التظليل إلى موقع الخانة التي يجري الفرز عليها فتتح كل لائحة من اللوائح التي تسبقها.

في الحواسيب التقليدية، وهي آلات ذات نفاذ عشوائي تسلسلي sequential random-access، نستخدم الفرز حسب الأساس أحياناً لفرز تسجيلات records من المعلومات ذات مفاتيح متعددة الحقول. على سبيل المثال، قد نرغب في فرز تواريخ من ثلاثة مفاتيح: السنة، والشهر، واليوم. يمكننا تنفيذ خوارزمية فرز مع دالة مقارنة بحيث إذا كان لدينا تاريخان فإننا نقارن الستين، فإذا تساوتا نقارن الشهرين، فإذا تساوتا أيضاً نقارن الأيام. بطريقة أخرى، يمكننا فرز للمعلومات ثلاث مرات باستخدام فرز مستقر: أولاً نفرز تبعاً للأيام، ثم تبعاً للشهر، وأخيراً تبعاً للسنة.

إن رماز الفرز حسب الأساس بسيط. نفترض الإجراءية التالية أن كل عنصر في الصفيفة  $A$  ذات الـ  $n$  عنصراً يتكوّن من  $d$  خانة، حيث الخانة 1 هي الخانة الدنيا مرتبة والخانة  $d$  هي العليا مرتبة.

**RADIX-SORT( $A, d$ )**

```
1 for  $i = 1$  to  $d$ 
2   use a stable sort to sort array  $A$  on digit  $i$ 
```

### توطئة 3.8

إذا كان لدينا  $n$  عدداً، كلٌّ منها يتكوّن من  $d$  خانة، وكلّ خانة يمكن أن تأخذ إحدى  $k$  قيمة ممكنة، فإن RADIX-SORT نفرز هذه الأعداد فرزاً صحيحاً في زمن  $\Theta(d(n+k))$  في حال كان الفرز المستقر الذي نستخدمه يستغرق زمناً  $\Theta(n+k)$ .

**البرهان:** نستنتج صحة الفرز حسب الأساس، بالاستقراء، على العمود الذي يجري فرزه (انظر التحرين 3-3.8). يعتمد تحليل زمن التنفيذ على الفرز المستقر المستخدم بوصفه خوارزمية فرز بسيطة. عندما تقع كل خانة في المجال من 0 إلى  $k-1$  (أي أنها يمكن أن تأخذ إحدى  $k$  قيمة ممكنة)، وقيمة  $k$  ليست جدّاً كبيرة، فإن الفرز بالعد يكون الخيار البديهي. يستغرق إذن كل مرور على  $n$  عدداً من  $d$  خانة زمناً  $\Theta(n+k)$ . ولما كان لدينا  $d$  مرورا، فإن الزمن الكلي للفرز حسب الأسس يساوي  $\Theta(d(n+k))$ . ■

إذا كان  $d$  ثابتاً، وكان  $k = O(n)$ ، يمكننا جعل الفرز حسب الأساس يُنفَّذ في زمن خطي. وفي الحالة الأعم، لدينا بعض المرونة في كيفية تفريق كل مفتاح إلى خانات.

#### 4.8 توطئة

إذا كان لدينا  $n$  عدداً، كلٌّ منها يتكوّن من  $b$  بتاً وأيّ عدد صحيح موجب  $r$ ، فإن RADIX-SORT تفرز هذه الأعداد فرزاً صحيحاً في زمن  $\Theta((b/r)(n + 2^r))$  إذا كان الفرز المستقر الذي يستخدمه يستغرق زمناً  $\Theta(n + k)$  في حال مُدخلات تقع ضمن المجال من 0 إلى  $k$ .

**البرهان:** يمكننا اعتبار أن كل مفتاح وكأنه يتكوّن من  $d = \lceil b/r \rceil$  خانة من  $r$  بتاً، لكل قيمة  $r$ . إن كل خانة هي عدد صحيح في المجال من 0 إلى  $2^r - 1$ ، ومنه يمكننا استخدام فرز بالعدّل لـ  $2^r - 1$ .  $k = 2^r - 1$ . (فمثلاً، يمكن اعتبار كلمة من 32 بتاً، وكأنها مكوّنة من أربع خانات كلٌّ منها مؤلّف من ثمانية بتات. إذن، لدينا  $32 = 2^5$ ، و  $r = 5$ ، و  $k = 2^5 - 1 = 255$ ، و  $d = b/r = 4$ ، يستغرق كل مرور للفرز بالعدّل زمناً  $\Theta(n + k) = \Theta(n + 2^r)$ . وحيث إنه لدينا  $d$  مروراً، فإن زمن التنفيذ الكلي يساوي  $\Theta(d(n + 2^r)) = \Theta((b/r)(n + 2^r))$ .

إذا كانت لدينا قيمتان معطتان  $n$  و  $b$ ، فكيف نختار القيمة  $r \leq b$  التي تجعل التعبير  $(b/r)(n + 2^r)$  أصغر؟ إذا كانت  $b < \lg n$ ، فإن  $\Theta(n) = \Theta(n + 2^r)$  لأية قيمة لـ  $r \geq b$ . وهكذا، إذا اخترنا  $r = b$  يكون زمن التنفيذ مساوياً  $\Theta(n) = \Theta((b/b)(n + 2^b))$ ، والتي هي أصغر بالمقارنة. وإذا كانت  $b \geq \lg n$ ، فإن اختيار  $r = \lg n$  يحقق أفضل زمن ضمن عامل ثابت، وهو ما سنراه هنا. وبذلك، فإن اختيارنا  $r = \lg n$  يجعل زمن التنفيذ من الرتبة  $\Theta(bn/\lg n)$ . فإذا زدنا  $r$  فوق  $\lg n$ ، فإن الحد  $2^r$  في البسط يزداد بسرعة أكبر من الحد  $r$  الموجود في المقام، ومن ثم فإن زيادة  $r$  فوق  $\lg n$  يجعل زمن التنفيذ من الرتبة  $\Omega(bn/\lg n)$ . أما إذا أنقصنا  $r$  عوضاً عن زيادتها، تحت  $\lg n$ ، فإن الحد  $b/r$  سيزداد، ويبقى الحد  $n + 2^r$  عند  $\Theta(n)$ .

هل خوارزمية الفرز حسب الأساس أفضل من خوارزميات الفرز التي تعتمد على المقارنة، كالفرز السريع مثلاً؟ إذا كانت  $b = O(\lg n)$ ، كما هو الحال غالباً، واخترنا  $r \approx \lg n$ ، فإن زمن تنفيذ الفرز حسب الأساس يكون  $\Theta(n)$ ، والذي يظهر أنه أفضل من توقع زمن الفرز السريع  $\Theta(n \lg n)$ . إلا أن المعاملات الثابتة للمقارنة في تدوين  $\Theta$  مختلفة. ومع أن الفرز حسب الأساس يمكن أن يمرّ (على  $n$  مفتاحاً) عدداً من المرات أقل من الفرز السريع، فإن كل مرور للفرز حسب الأساس يمكن أن يستغرق زمناً أطول بكثير من نظيره في الفرز السريع. لذا فإن اختيارنا لخوارزمية الفرز الفضلى يعتمد على مميزات التحيز، وعلى الحاسوب المستخدم (على سبيل المثال، غالباً ما يستخدم الفرز السريع عتاديات الذاكرة السريعة بفعالية أكبر من الفرز

حسب الأساس)، وعلى معطيات الدخل. يضاف إلى ذلك، أن نسخة الفرز حسب الأساس - التي تستخدم الفرز بالعد باعتباره فرعاً مستقرّاً وسيطاً - لا تفرز للمعطيات في المكان in-place، على حين أن كثيراً من طرق الفرز بالمقارنة التي تستغرق زمناً  $\Theta(n \lg n)$  تفرز للمعطيات في المكان. وبناء على ذلك، إذا احتلت ذاكرة التخزين الأولية للمقام الأول في الأهمية، يمكننا تفضيل خوارزمية فرز تُنفَّذ في المكان كالفرز السريع مثلاً.

## تمارين

### 1-3.8

استخدم الشكل 3.8 نموذجاً، واشرخ عقل RADIX-SORT على لائحة الكلمات الإنكليزية التالية: COW, DOG, SEA, RUG, ROW, MOB, BOX, TAB, BAR, EAR, TAR, DIG, BIG, TEA, NOW, FOX.

### 2-3.8

أتى من خوارزميات الفرز التالية مستقرة: الفرز بالإدراج، الفرز بالدمج، الفرز بالكومة، الفرز السريع؟ أعط آليّة scheme بسيطة تجعل أية خوارزمية فرز مستقرة. كم من الزمن الإضافي والذاكرة تستلزم آليتك؟

### 3-3.8

استخدم الاستقراء لتبرهن أن الفرز حسب الأساس يعمل كما يجب. أين يتطلب برهانك افتراض كون الفرز الوسيط مستقرّاً؟

### 4-3.8

بين كيف نفرز  $n$  عدداً صحيحاً في المجال من 0 إلى  $n^3 - 1$  في زمن  $O(n)$ .

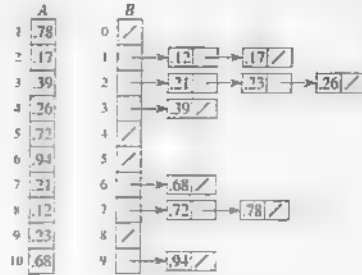
### \* 5-3.8

ما هو بالضبط عدد مرات الفرز اللازمة، في أول خوارزمية لفرز البطاقات وردت في هذا المقطع، وذلك لفرز أعداد عشرية من  $d$  خانة؟ وما هو عدد كدسات البطاقات التي قد يحتاج إليها العامل لربط البطاقات على نحو مستمر في أسوأ الحالات؟

## الفرز بالدلاء

4.8

يفترض الفرز بالدلاء bucket sort أن الدخل مستقر من توزيع منظم uniform distribution، وأن زمن تنفيذ هذا الفرز في الحالة الوسطى  $O(n)$ . إن الفرز بالدلاء سريع مثل الفرز بالعد، لأنه يفترض بعض الفرضيات على الدخل. ففي حين يفترض الفرز بالعد أن الدخل يتكون من أعداد صحيحة من مجال صغير، يفترض الفرز بالدلاء أن الدخل مؤلّد من إحصائية عشوائية تُوزّع العناصر توزيعاً منتظماً ومستقلاً في المجال  $[0, 1]$ . (انظر للمقطع 2، ففيه تعريف التوزيع المنتظم.)



**الشكل 4.8** تطبيق BUCKET-SORT في حالة  $n = 10$ . (أ) صفيقة الدخل  $A[1..10]$ . (ب) الصفيقة  $B[0..9]$  من لوائح مفروزة (دلاء) بعد السطر 8 للخوارزمية. يضم الدلو  $i$  القيم التي تقع في المجال نصف-المفتوح  $(i/10, (i+1)/10)$ . يتكون المخرج المفروز من ضم اللوائح  $B[0], B[1], \dots, B[9]$ ، الترتيب.

يُقسم الفرز بالدلاء المجال  $[0, 1]$  إلى  $n$  مجالاً جزئياً متساوياً أو دلاء *buckets*، ثم يوزع الدخل المتدخل في  $n$  عدداً في هذه الدلاء. ولما كان الدخل موزعاً توزيعاً منتظماً ومستقلاً في المجال  $[0, 1]$ ، فإننا لا نتوقع وقوع عدد كبير من الأعداد في كل دلو. وللحصول على المخرج، ما علينا إلا أن نفرز الأعداد في كل دلو، ثم نمرز على الدلاء بالترتيب، ونضع عناصر كل منها في لائحة.

يفترض رمزانا للفرز بالدلاء أن الدخل هو صفيقة  $A$  من  $n$  عنصر، وأن كل عنصر  $A[i]$  في الصفيقة يحقق المتراجحة  $0 \leq A[i] < 1$ . يحتاج الرمز إلى صفيقة مساعدة  $B[0..n-1]$  من لوائح مترابطة (دلاء) ويفترض وجود آلية للمحافظة على هذه اللوائح. (يصف المقطع 2.10 كيفية تنحيز العمليات الأساسية على اللوائح المترابطة.)

**BUCKET-SORT( $A$ )**

```

1   $n = A.length$ 
2  let  $B[0..n-1]$  be a new array
3  for  $i = 0$  to  $n-1$ 
4      make  $B[i]$  an empty list
5  for  $i = 1$  to  $n$ 
6      insert  $A[i]$  into list  $B[\lfloor n A[i] \rfloor]$ 
7  for  $i = 0$  to  $n-1$ 
8      sort list  $B[i]$  with insertion sort
9  concatenate the lists  $B[0], B[1], \dots, B[n-1]$  together in order
```

يبين الشكل 4.8 عملية الفرز بالدلاء على صفيقة دخل من 10 أعداد.

حتى نتأكد من صحة عمل هذه الخوارزمية، نتأمل العنصرين  $A[i]$  و  $A[j]$ . نفترض، دون أن يؤثر ذلك

على الحالة العامة، أن  $A[i] \leq A[j]$ ، لما كان  $A[i] \leq [nA[j]]$ ، فإن العنصر  $A[i]$  سيدخل إما ضمن الدلو نفسه مع  $A[j]$  أو في دلو دليله أقل. فإذا دخل  $A[i]$  و  $A[j]$  في الدلو نفسه، فإن حلقة for في السطرين 7-8 تضعهما في الترتيب الصحيح. وإذا دخلت  $A[i]$  و  $A[j]$  في دلوَين مختلفين، فإن السطر 9 يضعهما في الترتيب الصحيح. لذلك، فإن الفرز بالدلاء يعمل على نحو صحيح.

لتحليل زمن التنفيذ، لاحظ أن جميع الأسطر ما عدا السطر 8 يستغرق تنفيذها زمنًا  $O(n)$  في أسوأ الحالات. نحتاج إلى تحليل الزمن الكلي اللازم لاستدعاء فرز بالإدراج  $n$  مرة في السطر 8.

ولتحليل كلفة استدعاءات الفرز بالإدراج، نفترض أن  $n$  للتحوّل العشوائي الذي يشير إلى عدد العناصر الموضوعية في  $B[i]$ . ولما كان الفرز بالإدراج يُنفَّذ في زمنٍ تربيعي (انظر المقطع 2.2)، فإن زمن تنفيذ الفرز بالدلاء يساوي

$$T(n) = O(n) + \sum_{i=0}^{n-1} O(n_i^2) .$$

ندرس الآن زمن تنفيذ الفرز بالدلاء في الحالة الوسطى، وذلك بحساب قيمة التوقع لزمن التنفيذ، حيث نحسب التوقع اعتمادًا على التوزيع الاحتمالي للدخل. بأخذ توقع الطرفين وباستخدام خطية التوقع، نحصل على

$$\begin{aligned} E[T(n)] &= E\left[O(n) + \sum_{i=0}^{n-1} O(n_i^2)\right] \\ &= O(n) + \sum_{i=0}^{n-1} E[O(n_i^2)] \quad (\text{اعتمادًا على خطية التوقع}) \\ &= O(n) + \sum_{i=0}^{n-1} O(E(n_i^2)) \quad ((2.2) \text{ اعتمادًا على المعادلة}) \end{aligned} \quad (1.8)$$

نَدَّعي أنَّ

$$E[n_i^2] = 2 - 1/n \quad (2.8)$$

لقيم  $i = 0, 1, \dots, n-1$ . ليس من المفاجئ أن يكون لكل دلو  $i$  القيمة نفسها لـ  $E[n_i^2]$ ، لأن كل قيمة في صفيحة الدخل  $A$  تقع باحتمال متساوٍ في أي دلو. ولبرهان للمعادلة (2.8) نُعرِّف للتحويلات العشوائية المؤشرة (انظر المقطع 2.5)

$$X_{ij} = I\{A[i] \text{ falls in bucket } i\}$$

حيث  $i = 0, 1, \dots, n-1$  و  $j = 1, 2, \dots, n$ ، ومنه،

$$n_i = \sum_{j=1}^n X_{ij}$$



ولحساب  $E[n_f^2]$ ، فإننا ننشر التوزيع ونعيد تجميع الحدود:

$$\begin{aligned}
 E[n_f^2] &= E\left[\left(\sum_{j=1}^n X_{ij}\right)^2\right] \\
 &= E\left[\sum_{j=1}^n \sum_{k=1}^n X_{ij} X_{ik}\right] \\
 &= E\left[\sum_{j=1}^n X_{ij}^2 + \sum_{1 \leq j \leq n} \sum_{\substack{1 \leq k \leq n \\ k \neq j}} X_{ij} X_{ik}\right] \\
 &= \sum_{j=1}^n E[X_{ij}^2] + \sum_{1 \leq j \leq n} \sum_{\substack{1 \leq k \leq n \\ k \neq j}} E[X_{ij} X_{ik}] . \quad (3.8)
 \end{aligned}$$

حيث ينتج السطر الأخير من خطية التوقع. سوف نحسب كل مجموع على حدة. إن المتحول العشوائي المؤشر  $X_{ij}$  يساوي 1 باحتمال  $1/n$  و 0 ما عدا ذلك، ولذا يكون لدينا

$$\begin{aligned}
 E[X_{ij}^2] &= 1^2 \times \frac{1}{n} + 0^2 \times \left(1 - \frac{1}{n}\right) \\
 &= \frac{1}{n} .
 \end{aligned}$$

عندما  $f \neq k$ ، فإن المتحولين  $X_{ij}$  و  $X_{ik}$  مستقلان، ومنه

$$\begin{aligned}
 E[X_{ij} X_{ik}] &= E[X_{ij}] E[X_{ik}] \\
 &= \frac{1}{n} \times \frac{1}{n} \\
 &= \frac{1}{n^2} .
 \end{aligned}$$

بتعويض هاتين القيمتين للتوقعتين في المعادلة (3.8)، نحصل على

$$\begin{aligned}
 E[n_f^2] &= \sum_{j=1}^n \frac{1}{n} + \sum_{1 \leq j \leq n} \sum_{\substack{1 \leq k \leq n \\ k \neq j}} \frac{1}{n^2} \\
 &= n \times \frac{1}{n} + n(n-1) \times \frac{1}{n^2} \\
 &= 1 + \frac{n-1}{n} \\
 &= 2 - \frac{1}{n} .
 \end{aligned}$$

وهذا ما يبرهن المعادلة (2.8).

باستخدام قيمة التوقع في المعادلة (1.8)، نستنتج أن زمن تنفيذ الفرز بالدلاء في الحالة الوسطى يساوي

$$\Theta(n) \approx \Theta(n) + n \times O(2 - 1/n).$$

إن الفرز بالدلاء لا يزال ممكن التنفيذ في زمن خطي، ولو كان الدخول غير ناتج عن توزيع منتظم. فمادام الدخول يتمتع بخاصية كون مجموع مربعات أحجام الدلاء خطيًا مع العدد الكلي للعناصر، فإن المعادلة (1.8) تبين أن الفرز بالدلاء سوف ينفَّذ في زمن خطي.

## تمارين

### 1-4.8

استخدم الشكل 4.8 نموذجًا، لشرح عمل BUCKET-SORT على الصيغة

$$A = \{.79, .13, .16, .64, .39, .20, .89, .53, .71, .42\}$$

### 2-4.8

علّل لماذا يكون زمن تنفيذ خوارزمية الفرز بالدلاء في أسوأ الحالات  $\Theta(n^2)$ ؟ كيف يمكن أن نجري تعديلًا بسيطًا على الخوارزمية بحيث يحافظ على زمن التنفيذ في الحالة الوسطى ويجعل زمن التنفيذ في أسوأ الحالات  $\Theta(n \lg n)$ ؟

### 3-4.8

ليكن  $X$  متحولًا عشوائيًا يساوي عدد مرات الحصول على وجه heads في رميتين لقطعة نقد عادلة. ما هي قيمة  $E[X^2]$  وما هي قيمة  $E^2[X]$ ؟

### 4-4.8

ليكن لدينا  $n$  نقطة ضمن الدائرة الواحدة  $p_i = (x_i, y_i)$  بحيث يكون  $0 < x_i^2 + y_i^2 \leq 1$  لكل  $i = 1, 2, \dots, n$ . افترض أن النقاط موزعة بانتظام؛ أي إن احتمال وجود نقطة في أية منطقة من الدائرة يتناسب مع مساحة تلك المنطقة. صمّم خوارزمية يكون زمن تنفيذها في الحالة الوسطى  $\Theta(n)$  لفرز  $n$  نقطة تبعًا لبعدها عن المركز  $d_i = \sqrt{x_i^2 + y_i^2}$  عن المركز. (تلميح: صمّم أحجام الدلاء في BUCKET-SORT كي يظهر التوزيع المنتظم للنقاط في الدائرة الواحدة).

### \* 5-4.8

تُعرف دالة التوزيع الاحتمالي *probability distribution function*  $P(x)$  لتحول عشوائي  $X$  بالعلاقة  $P(x) = \Pr\{X \leq x\}$ . افترض أننا سحبنا لائحة من  $n$  متحولًا عشوائيًا  $X_1, X_2, \dots, X_n$  تتبع دالة توزيع احتمالي مستمر  $P$  قابل للحساب في زمن  $O(1)$ . بيّن كيف نفرز هذه الأعداد في زمن توقعه خطي في الحالة الوسطى.

## مسائل

## 1-8 حدود احتمالية دنيا على الفرز بالمقارنة

نبرهن في هذه المسألة، أن  $\Omega(n \lg n)$  يمثل حدًا أدنى احتماليًا لزمن تنفيذ أي فرز بالمقارنة حتمي أو deterministic أو ذي عشوائية مضافة randomized على  $n$  عنصرٍ دخلٍ متمايزًا. تبدأ بدراسة فرز بالمقارنة حتمي  $A$  مع شجرة قرار  $T_A$ . ونفترض أن كل تبديل لمدخلات  $A$  له الاحتمال نفسه.

أ. افترض أن كل ورقة من  $T_A$  ألصق بها احتمال بلوغها دخلًا عشوائيًا ما. برهن أن  $n!$  ورقة تمامًا سوف يُلصق بها  $1/n!$  ويلصق بالباقي 0.

ب. نرمز بـ  $D(T)$  إلى طول المسار الخارجي لشجرة قرار  $T$ ؛ أي إن  $D(T)$  يساوي مجموع أعماق جميع أوراق  $T$ . ولتكن  $T$  شجرة قرار عدّد أوراقها  $k > 1$ ، ولتكن  $LT$  و  $RT$  الشجرتين الفرعيتين اليسرى واليمنى لـ  $T$ . بيّن أن  $D(T) = D(LT) + D(RT) + k$ .

ث. ليكن  $d(k)$  القيمة الصغرى لـ  $D(T)$  على جميع أشجار القرار  $T$  التي عدد أوراقها  $k > 1$ . بيّن أن  $d(k) = \min_{1 \leq i \leq k-1} \{d(i) + d(k-i) + k\}$ . (نلمح: فكّر في شجرة قرار  $T$  عدد أوراقها  $k$  تحقق القيمة الصغرى. وافترض أن  $i_0$  عدد الأوراق في  $LT$  و  $k - i_0$  عدد الأوراق في  $RT$ .)

ث. برهن أن الدالة  $i \lg i + (k-i) \lg(k-i)$  أصغر عند  $i = k/2$ ، حيث  $k > 1$  و  $i$  في المجال  $1 \leq i \leq k-1$ . استنتج أن  $d(k) = \Omega(k \lg k)$ .

ج. برهن أن  $D(T_A) = \Omega(n! \lg(n!))$ ، واستنتج أن زمن التنفيذ في الحالة الوسطى لفرز  $n$  عنصرًا هو  $\Omega(n \lg n)$ .

لنناقش، الآن، خوارزمية فرز بالمقارنة ذات عشوائية مضافة  $B$ . يمكننا توسيع نموذج شجرة القرار ليتعامل مع العشوائية المضافة، وذلك بتعريف نوعين من العقد: عقد مقارنة اعتيادية وعقد "إضافة عشوائية randomization". نُمذج عُقد الإضافة العشوائية الخياز العشوائي من الشكل  $\text{RANDOM}(1, r)$  الذي تستخدمه الخوارزمية  $B$ ؛ حيث يكون للعقدة  $r$  ابنًا، يُمكن اختيار أيٍّ منهم باحتمال متساوٍ خلال تنفيذ الخوارزمية.

ح. بيّن أنه في أي فرز بالمقارنة ذي عشوائية مضافة  $B$ ، يوجد فرز حتمي بالمقارنة  $A$  لا يتجاوز عدّد المقارنات المتوقع مثله في الفرز  $B$ .

## 2-8 الفرز في المكان في زمن خطي

افترض أننا نريد فرز صليغة من  $n$  تسجيلة معطيات، وأن مفتاح كل تسجيلة له القيمة 0 أو 1. لعلّ خوارزمية

فرز مثل هذه التسجيلات تمتلك بعضًا من المميزات الثلاث التالية المرغوب فيها:

1. تُنفذ الخوارزمية في زمن  $O(n)$ .
2. الخوارزمية مستقرة.
3. تفرز الخوارزمية في المكان، مستخدمة قدرًا ثابتًا وليس أكثر من مساحة التخزين إضافة إلى الصيغة الأصلية.

- أ. أعط خوارزمية تحقق للمعيارين 1 و 2.
- ب. أعط خوارزمية تحقق للمعيارين 1 و 3.
- ت. أعط خوارزمية تحقق للمعيارين 2 و 3.
- ث. هل يمكنك استخدام أي من خوارزميات الفرز التي اقترحتها في (أ) إلى (ت) طريقة للفرز المستخدم في السطر 2 في RADIX-SORT، بحيث تفرز RADIX-SORT  $n$  تسجيلة تتكون مفاتيحها من  $b$  بتًا في زمن  $O(bn)$ ؟ اشرح كيف أو لم لا.

- ج. افترض أن مفاتيح  $n$  تسجيلة تقع في المجال من 1 إلى  $k$ . بيّن كيف يمكن تعديل الفرز بالعد بحيث تُفرز التسجيلات في المكان في زمن  $O(n+k)$ . يمكنك استخدام مساحة تخزين  $O(k)$  إضافة إلى صيغة الدخول. هل خوارزمتك مستقرة؟ (تلميح: ما الذي كنت ستفعله في حالة  $k = 3$ ؟)

### 3-8 فرز عناصر ذات أطوال مختلفة

- أ. ليكن لدينا صيغة من أعداد صحيحة، حيث يمكن أن يكون للأعداد الصحيحة المختلفة عددٌ مختلفٌ من الحانات، إلا أن عدد الحانات الكلي لكل الأعداد الصحيحة في الصيغة يساوي  $n$ . بيّن كيف يمكن فرز الصيغة في زمن  $O(n)$ .
  - ب. ليكن لديك صيغة من متتاليات محارف، حيث يمكن أن يكون لمتتاليات المحارف المختلفة عددٌ مختلفٌ من المحارف، إلا أن عدد المحارف الكلي في جميع متتاليات المحارف يساوي  $n$ . بيّن كيف يمكن فرز متتاليات المحارف في زمن  $O(n)$ .
- (انتبه إلى أن الترتيب المرغوب فيه هنا هو الترتيب الأبجدي للتعارف؛  $a < ab < b$ ، مثلاً.)

### 4-8 أباريق الماء

افترض أن لديك  $n$  إبريقًا أحمر و  $n$  إبريقًا أزرق، جميعها مختلفة في الشكل والحجم. تنسع الأباريق الأحمر كميات مختلفة من الماء، كما هو حال الأباريق الزرقاء. إضافة إلى ذلك، يوجد لكل إبريق أحمر إبريق أزرق

يتسع كمية الماء نفسها، والعكس بالعكس.

تتمكن مهنتك في تجميع الأباريق في أزواج من الأباريق الحمراء والزرقاء المتساوية السعة. لتحقيق ذلك، يمكنك القيام بالعملية التالية: اختر زوجاً من الأباريق أحدهما أحمر والآخر أزرق. امسك الإبريق الأحمر بالماء، ثم أفرغ هذا الماء في الإبريق الأزرق. ستعلم من هذه العملية أنّها من الإبريقين الأحمر أم الأزرق يمكن أن يتسع إلى كمية مياه أكبر، أو إن كان لهما الحجم نفسه. افترض أن عملية المقارنة هذه تستغرق وحدة زمنية واحدة. تتمكن مهنتك في إيجاد خوارزمية تنفذ عدداً أصغرياً من المقارنات لتحديد المجموعات. تذكر أنك لا تستطيع مقارنة إبريقين أحمرين أو إبريقين أزرقين مباشرة.

أ. صنف خوارزمية حتمية تستخدم  $\Theta(n^2)$  عملية مقارنة لتجميع الأباريق في أزواج.

ب. برهن أن الحد الأدنى لعدد المقارنات التي يجب أن تُنفذها الخوارزمية لحل هذه المسألة هو  $\Omega(n \lg n)$ .

ت. أعط خوارزمية ذات عشوائية مضافة يكون عدد المقارنات المتوقع فيها هو  $O(n \lg n)$ ، وبرهن أن هذا الحد صحيح. ما هو عدد المقارنات في أسوأ الحالات في خوارزمتك؟

### 5-8 الفرز الوسيط

افترض أننا عوضاً عن فرز صفيقة، نريد فقط أن نترتيب العناصر في المتوسط. وبعبارة أدق، نقول عن صفيقة  $A$  من  $n$  عنصراً إنها  $k$ -مرتبة  $k$ -sorted إذا كانت العلاقة التالية محققة لكل قيم  $i = 1, 2, \dots, n - k$

$$\frac{\sum_{j=i}^{i+k-1} AU_j}{k} \leq \frac{\sum_{j=i+1}^{i+k} AU_j}{k}.$$

أ. ماذا يعني أن تكون الصفيقة 1-مرتبة.

ب. أعط تبديلاً للأعداد  $1, 2, \dots, 10$  بحيث تكون 2-مرتبة، دون أن تكون مرتبة.

ت. برهن أن صفيقة من  $n$  عنصراً تكون  $k$ -مرتبة إذا وفقط إذا كان  $A[i] \leq A[i + k]$  لكل قيم  $i = 1, 2, \dots, n - k$ .

ث. أعط خوارزمية تحمل صفيقة من  $n$  عنصراً  $k$ -مرتبة في زمن  $O(n \lg(n/k))$ .

يمكننا أيضاً إيجاد حدّ أدنى للزمن اللازم لإنتاج صفيقة  $k$ -مرتبة، عندما تكون  $k$  ثابتة.

ج. بيّن أنه يمكننا فرز صفيقة  $k$ -مرتبة طولها  $n$  في زمن  $O(n \lg k)$ . (لمسح: استخدم حل التمرين 5.6-9.)

ح. بيّن أنه عندما تكون  $k$  ثابتة، نحتاج إلى زمن  $\Omega(n \lg n)$  لحمل صفيقة من  $n$  عنصراً  $k$ -مرتبة.

(لمسح: استخدم حل الجزء السابق مع الحد الأدنى للفرز بالمقارنة.)

### 6-8 الحد الأدنى للدمج لوائح مرتبة

تبرز مسألة دمج لائحتين مرتبتين كثيرًا. وقد رأينا في المقطع 1-3.2 إجرائية لدمج لائحتين مرتبتين بوصفها مسألاً فرعياً يُدعى MERGE. سنُبرهن في هذه المسألة أن حدًا أدنى يساوي  $2n - 1$  لعدد المقارنات اللازمة في أسوأ الحالات لدمج لائحتين مرتبتين، تضم كل منهما  $n$  عنصرًا. سنبدأ أولاً، حدًا أدنى للمقارنات  $2n - o(n)$  باستخدام شجرة قرار.

أ. ليكن لدينا  $2n$  عددًا، احسب عدد الطرق الممكنة لتقسيم العناصر إلى لائحتين مرتبتين، تضم كل منهما  $n$  عنصرًا.

ب. بين، باستخدام شجرة قرار وجوابك عن السؤال (أ)، أن أية خوارزمية تدمج لائحتين مرتبتين دمجًا صحيحًا يجب أن تُنجز على الأقل  $2n - o(n)$  مقارنة.

نبين الآن حدًا أكثر ملاصقة نوعًا ما وهو  $2n - 1$ .

ت. بين أنه إذا كان عنصران متتاليان في ترتيب مفروز هما من لائحتين مختلفتين، فلا بد أنه قد جرت مقارنتهما.

ث. استخدم إجابتك عن الجزء السابق لتبين أن  $2n - 1$  هو حد أدنى للمقارنات لدمج لائحتين مرتبتين.

### 7-8 توطئة الفرز 0-1 وخوارزمية columnsort

نأخذ عملية قارن-بازل *compare-exchange* على عنصرَي صغيفة  $A[i]$  و  $A[j]$ ، حيث  $j < i$ ، الصيغة التالية:

COMPARE-EXCHANGE( $A, i, j$ )

```
1 if  $A[i] > A[j]$ 
2   exchange  $A[i]$  with  $A[j]$ 
```

نعلم، بعد تنفيذ العملية قارن-بازل، أن  $A[i] \leq A[j]$ .

نعمل خوارزمية قارن-بازل *مغفلة oblivious compare-exchange algorithm* فقط على متتالية مُحددة سلفًا من عمليات قارن-بازل. يجب أن تكون أدلة المواقع المُراد مقارنتها في المتتالية محددة سلفًا، ومع أن هذه الأدلة قد تعتمد على عدد العناصر المفروزة، إلا أنه لا يمكنها أن تعتمد على القيم المفروزة، ولا على نتيجة أية عملية قارن-بازل سابقة. على سبيل المثال، نُعبّر هنا عن فرز بالإدراج باعتباره خوارزمية قارن-بازل مغفلة:

INSERTION-SORT( $A$ )

```
1 for  $j = 2$  to  $A.length$ 
2   for  $i = j - 1$  downto 1
3     COMPARE-EXCHANGE( $A, i, i + 1$ )
```

تقدم **توطئة الفرز 0-1 sorting lemma** طريقة فعالة للبرهان على أن خوارزمية قارن-بادل مغلفة تقدم نتيجة مفروزة. تنص هذه التوطئة على أنه إذا كانت خوارزمية قارن-بادل المغلفة تفرز فرزًا صحيحًا كلَّ متتالية دخل مكونة من أصغارٍ ووحدانٍ فقط، فإنها تفرز فرزًا صحيحًا كلَّ المُدخلات مهما تكن قيمها. المطلوب برهان توطئة الفرز 0-1 بطريقة البرهان للعكس الموجب *contrapositive*: إذا أخفقت خوارزمية قارن-بادل مغلفة في فرز دخل اعتباطي، فسُتُحقق في فرز دخل ما من الأصغار والوحدات. افترض أن خوارزمية قارن-بادل مغلفة أخفقت في فرز الصيغة  $A[1..n]$  فرزًا صحيحًا. لنكن  $A[p]$  أصغر قيمة في  $A$  تضعها الخوارزمية  $X$  في المكان الخاطئ، ولنكن  $A[q]$  القيمة التي تضعها الخوارزمية  $X$  في المكان الذي من المفترض أن تذهب إليه  $A[p]$ . عرّف صيغة  $B[1..n]$  من الأصغار والوحدات كما يلي:

$$B[i] = \begin{cases} 0 & \text{if } A[i] \leq A[p] \\ 1 & \text{if } A[i] > A[p] \end{cases}$$

أ. ناقش أنه إذا كان  $A[q] > A[p]$ ، فإن  $B[q] = 1$  و  $B[p] = 0$ .

ب. لإثبات برهان توطئة الفرز 0-1، برهن أن الخوارزمية  $X$  تُحقق في فرز الصيغة  $B$  فرزًا صحيحًا.

سوف نستخدم الآن توطئة الفرز 0-1 للبرهان على أن خوارزمية فرز معينة تعمل بوجهٍ صحيح. تعمل الخوارزمية **columnsort** على صيغة مستطيلة من  $n$  عنصرًا. تتكون الصيغة من  $r$  سطرًا و  $s$  عمودًا (أي  $n = rs$ )، ونخضع لثلاثة قيود:

- يجب أن يكون  $r$  زوجيًا،
- يجب أن تقبل  $r$  القسمة على  $s$ ،
- $r \geq 2s^2$ .

عند انتهاء **columnsort**، تكون الصيغة **مفروزة وفق أعمدها** *column-major order*: أي إذا قرأنا الأعمدة نزولًا، من اليسار إلى اليمين، تكون العناصر متزايدة باطراد. تتبع **columnsort** ثماني خطوات، بقطع النظر عن قيمة  $n$ . الخطوات الفردية كلها متماثلة: افرز كل عمود على حدة. تتمز كل خطوة زوجية تبديلاً محددًا. وهذه الخطوات هي:

1. افرز كل عمود.
2. انقل **transpose** الصيغة، ولكن مع إعادة تشكيلها لتصبح  $r$  سطرًا و  $s$  عمودًا. بعبارة أخرى، ضع العمود الأيسر الأول في الأسطر  $r/s$  العليا، بالترتيب؛ ثم ضع العمود التالي في الأسطر  $r/s$  التالية، بالترتيب، وهكذا ...
3. افرز كل عمود.

1 4 11	1 3 6	4 8 10	4 1 2	10 14 5
3 8 14	2 5 7	12 ■ 18	8 3 5	8 7 17
6 10 17	4 8 10	1 3 7	10 7 6	12 1 6
2 9 12	9 13 15	9 14 15	12 9 11	16 9 11
5 13 16	11 14 17	2 5 6	16 14 13	4 15 2
7 15 18	12 16 18	11 13 17	18 15 17	18 3 13
(ج)	(ث)	(ت)	(ب)	(أ)

1 7 13	4 10 16	5 10 16	1 4 11
2 8 14	5 11 17	6 13 17	2 8 12
3 9 15	6 12 18	7 15 18	3 9 14
4 10 16	1 7 13	1 4 11	5 10 16
5 11 17	2 8 14	2 8 12	■ 13 17
6 12 18	3 9 15	3 9 14	7 15 18
(د)	(ذ)	(خ)	(ح)

**الشكل 5-8** خطوات columnsort. (أ) صيغة الدخول من 6 أسطر و 3 أعمدة. (ب) بعد فرز كل عمود في الخطوة 1. (ت) بعد عملية النقل وإعادة التشكيل في الخطوة 2. (ث) بعد فرز كل عمود في الخطوة 3. (ج) بعد تنفيذ الخطوة 4، التي تنجز عكس التبديل المُنفَّذ في الخطوة 2. (ح) بعد فرز كل عمود في الخطوة 5. (خ) بعد إزاحة بمقدار نصف عمود في الخطوة 6. (ذ) بعد فرز كل عمود في الخطوة 7. (د) بعد تنفيذ الخطوة 8، التي تنجز عكس التبديل المُنفَّذ في الخطوة 6. الصيغة الآن مفروزة وفق أعمدها.

4. أُنجز عكس التبديل المُنفَّذ في الخطوة 2.
5. افرز كل عمود.
6. انقل النصف الأعلى من كل عمود إلى النصف الأسفل من العمود نفسه، وانقل النصف الأسفل من كل عمود إلى النصف الأعلى من العمود الذي يليه. أترك النصف الأعلى من العمود الأخير الأول فارغاً. انقل النصف الأسفل من العمود الأخير إلى النصف الأعلى من عمود أمين أعير جديد تنشئه، وأترك النصف الأسفل من هذا العمود الجديد فارغاً.
7. افرز كل عمود.
8. أُنجز عكس التبديل المُنفَّذ في الخطوة 6.

يبين الشكل 5.8 مثالاً على خطوات columnsort في حالة  $r = 6$  و  $s = 3$ . (ومع أن هذا المثال لا يحقق المطلب  $r \geq 2s^2$ ، فإنه يعمل.)

ت. يَبَيِّن أنه يمكن معالجة columnsort كخوارزمية قارن-بادل مغلقة، ولو كنا لا نعلم طريقة الفرز التي نستخدمها الخطوات الفردية.

وعلى الرغم من أنه قد يبدو من الصعب التصديق أن columnsort تفرز حقاً، فإنك ستستعجب توطئة



الفرز 0-1 للرهان على قيامها بذلك. من الممكن تطبيق توطئة الفرز 0-1 لكوننا نستطيع معاملة columnsort كنوازيمة قارن-بازل مغلقة. سوف يساعدك التعريفان التاليان في تطبيق توطئة الفرز 0-1. نقول عن منطقة في صيغة أنها نظيفة *clean* إذا كنا نعلم أنها تحتوي إما أصفًا فقط وإما وحدانيًا فقط. فإذا كانت المنطقة تحتوي مزيجًا من الأصفار والوحدات، فهي ملوثة *dirty*. افترض، من الآن فصاعدًا، أن الصيغة تحتوي أصفًا ووحيدانيًا فقط، وأنه يمكننا معاملة كصيغة من  $s$  سطرًا و  $s$  عمودًا.

ث. برهن أن الصيغة، بعد الخطوات 3-1، تضم بعض الأسطر النظيفة في أعلاها، وبعض الأسطر النظيفة من الوحدات في أسفلها، و  $s$  سطرًا ملوثًا بينهما على الأكثر.

ج. برهن أنه إذا قرأنا أعمدة الصيغة واحدًا تلو الآخر، بعد الخطوة 4، فإننا نبدأ بمنطقة نظيفة من الأصفار، وننتهي بمنطقة نظيفة من الوحدات، وتحتل منطقة ملوثة لا تتجاوز  $s^2$  عنصرًا في الوسط.

ح. برهن أن الخطوات 8-5 تنتج خرجًا مقروءًا كليًا من الأصفار والوحدات. استنتج أن columnsort تفرز فرزًا صحيحًا كل المدخلات مهما تكن قيمها.

د. افترض الآن أن  $s$  لا يقبل القسمة على  $s$ . برهن أنه بعد الخطوات 3-1، تحتوي الصيغة على بعض الأسطر النظيفة من الأصفار في القمة، وبعض الأسطر النظيفة من الوحدات في الأسفل، وعلى الأكثر  $1 \sim 2s$  سطرًا ملوثًا بينهما. ما هو مقدار كثير  $s$  بالنسبة إلى  $s$  اللازم حتى تفرز columnsort فرزًا صحيحًا عندما لا تقبل  $s$  القسمة على  $s$ ؟

ذ. اقترح تغييرًا بسيطًا للخطوة 1 بحيث نسمح لنا بالحفاظ على المطلوب  $2s^2 \geq s$  حتى عندما لا تقبل  $s$  القسمة على  $s$ ، برهن على أن columnsort تفرز فرزًا صحيحًا، بوجود تعديلك.

## ملاحظات الفصل

كان Ford و Johnson [110] أول من استخدم نموذج شجرة القرار لدراسة الفرز بالمقارنة. يتناول البحث الشامل ل Knuth [211] في الفرز أشكالاً عديدة مختلفة لمسألة الفرز، ومنها الحد الأدنى النظري information-theoretic لتعقيد الفرز الذي تعرضنا له هنا. ذُرس Ben-Or [39] حدودًا دنيا للفرز باستخدام تعميمات لنموذج شجرة القرار.

يسبب Knuth إلى H. H. Seward اختراع الفرز بالعد في عام 1945، وكذلك فكرة الجمع بين الفرز بالعد والفرز حسب الأساس. يبدو أن الفرز حسب الأساس الذي يبدأ بالخانة الأقل قيمة كان خوارزمية شعبية استخدمها مشغلو فارزات البطاقات الميكانيكية على نطاق واسع. تبعًا ل Knuth، فإن أول إشارة

منشورة لهذه الطريقة هي وثيقة لـ L. J. Comrie عام 1929 تصف معدات للبطاقات المثقبة. تُستخدم خوارزمية الفرز بالدلاء منذ عام 1956، عندما اقترح الفكرة الأساسية R. C. Singleton و E. J. Isaac [188].

قدّم Munro و Raman [263] خوارزمية فرز مستقرة تُنتج  $O(n^{1+\epsilon})$  مقارنةً في أسوأ الحالات، حيث  $0 < \epsilon \leq 1$  أي ثابت محدد. ومع أن كل الخوارزميات التي تُنفَّذ في زمن  $O(n \lg n)$  تقوم بعدد أقل من المقارنات، فإن خوارزمية Munro و Raman تُحرك المعطيات  $O(n)$  مرةً فقط، وهي تعمل في المكان.

دُرِسَ العديد من الباحثين حالة فرز  $n$  عددًا صحيحًا من  $b$  بتًا في زمن  $O(n \lg n)$ . وقد جرى الوصول إلى العديد من النتائج الإيجابية، كل منها تعتمد على فرضيات مختلفة قليلًا بالنسبة إلى نموذج الحوسبة والقيود المفروضة على الخوارزمية. نفترض كل النتائج أن ذاكرة الحاسوب مقسمة إلى كلمات من  $b$  بتًا قابلة للعنونة.

قدّم Fredman و Willard [115] بنية معطيات شجرة صهر fusion tree واستخدامها لفرز  $n$  عددًا صحيحًا في زمن  $O(n \lg n / \lg \lg n)$ . وحسّن Andersson [16] هذا الحد لاحقًا إلى  $O(n \sqrt{\lg n})$ . نحتاج هذه الخوارزميات إلى استخدام عمليات جداء والعديد من الثوابت المحسوبة سلفًا. بيّن كلٌّ من Andersson و Hagerup و Nilsson و Raman [17] كيف يمكن فرز  $n$  عددًا صحيحًا في زمن  $O(n \lg \lg n)$  دون استخدام عمليات جداء، ولكن طريقتهم تتطلب ذاكرة تخزين يمكن أن تكون غير محدودة بدلالة  $n$ . يمكن، باستخدام تليد جدائي multiplicative hashing، إنقاص حجم ذاكرة التخزين اللازمة إلى  $O(n)$ ، ولكن يصبح الحد  $O(n \lg \lg n)$  في أسوأ حالات زمن التنفيذ الوسطي حدًا الزمن المتوقع expected-time bound.

قدّم Thorup [335] نعيمًا لأشجار البحث الأسية لـ Andersson [16]، بخوارزمية فرز في زمن  $O(n (\lg \lg n)^2)$  لا تستخدم الجداء ولا عشوائية مضافة، وتستخدم مساحة تخزين خطية. وحسّن Han [158] حدّ الفرز إلى  $O(n \lg \lg n \lg \lg \lg n)$ ، وذلك بمزج هذه التقنيات مع بعض الأفكار الجديدة. ومع أن هذه الخوارزميات تمثل قفزات نظرية ذات أهمية، لكنها جميعًا معقدة إلى حدٍّ ما، ومن المستبعد في الوقت الحالي أن تنافس خوارزميات الفرز الراهنة المستخدمة.

تُنسب خوارزمية columnsort في المسألة 7-8 إلى Leighton [227].

## 9 الأوساط وإحصائيات الترتيب

إن **إحصائية الترتيب  $i$**  ( *$i$ th order statistic*) مجموعة من  $n$  عنصرًا هي العنصر الأصغر ذو الترتيب  $i$ . على سبيل المثال، إن الأصغر *minimum* مجموعة من العناصر هو إحصائية الترتيب الأولى ( $i = 1$ )، والأكبر *maximum* هو إحصائية الترتيب  $n$  ( $i = n$ ). والوسط *median*، على نحو غير صوري، هو النقطة في "الوسط" لمجموعة. فإذا كان  $n$  فرديًا، فإن الوسط وحيد، ويقع عند  $(n + 1)/2$ . وإذا كان  $n$  زوجيًا، فلدينا وسطان يقعان عند  $n/2$  و  $n/2 + 1$ . وهكذا، ويقطع النظر عن ازدواجية (ندية parity)  $n$ ، فإن الوسطين يقعان عند  $i = \lfloor (n + 1)/2 \rfloor$  (الوسط الأدنى *lower median*) و  $i = \lceil (n + 1)/2 \rceil$  (الوسط الأعلى *upper median*). وللتبسيط، سنستخدم في هذا الكتاب، على الدوام، التعبير "الوسط *the median*" للإشارة إلى الوسط الأدنى.

يناقش هذا الفصل مسألة اختبار إحصائية الترتيب  $i$  لمجموعة من  $n$  عددًا متمايزًا. وسنفترض للملاءمة أن المجموعة تحتوي أعدادًا متمايزة، مع أننا فعليًا يمكننا تعميم كل شيء نقوم به على الحالة التي تحتوي فيها المجموعة قيمًا مكررة. يمكننا تحديد **مسألة الاختيار *selection problem*** صوريًا كما يلي:

**الدخل:** مجموعة  $A$  من  $n$  عددًا (متمايزًا) وعدد صحيح  $i$ ، حيث  $1 \leq i \leq n$ .

**الخروج:** العنصر  $x \in A$  الذي هو أكبر ثمانًا من  $i - 1$  عنصرًا سواه في المجموعة  $A$ .

يمكننا حل مسألة الاختيار بزم  $O(n \lg n)$ ، حيث يمكننا فرز الأعداد باستخدام الفرز بالكومة أو بالدمج، ثم نشير ببساطة إلى العنصر ذي الدليل  $i$  في صيغة الخرج. يعرض هذا الفصل خوارزميات أسرع.

نتناقش، في المقطع 1.9، مسألة اختبار أصغر عنصر في مجموعة من العناصر وأكبر عنصر فيها. وتُعتبر مسألة الاختيار العامة هي المسألة الأكثر أهمية التي سندرسها في المقطعين التاليين. يُحلل المقطع 2.9 خوارزمية ذات عشوائية مضافة تُحقق زمن تنفيذ متوقع  $O(n)$ ، وذلك بافتراض أن العناصر متمايزة. ويتضمن المقطع 3.9 خوارزمية ذات أهمية نظرية كبيرة تُحقق زمن تنفيذ  $O(n)$  في أسوأ الحالات.

## 1.9 الأصغر والأكبر

ما هو عدد المقارنات اللازمة لتحديد العنصر الأصغر في مجموعة من  $n$  عنصرًا؟ يمكننا بسهولة الحصول على حدّ أعلى لـ  $n-1$  عملية مقارنة: نفحص كل عنصر من المجموعة بالترتيب (واحدًا تلو الآخر) ونحتفظ بأثر أصغر عنصر جرى العثور عليه آنفًا. سنفترض، في الإجراء التالي، أن المجموعة موجودة في صيغة  $A$  طولها  $n$   $A.length = n$ .

```

MINIMUM(A)
1  min = A[1]
2  for i = 2 to A.length
3      if min > A[i]
4          min = A[i]
5  return min

```

يمكننا بالطبع، أيضًا، إيجاد أكبر عنصر بإجراء  $n-1$  مقارنة. ولكن، هل هذا أقصى ما نستطيع فعله؟ الجواب: نعم، لأنه يمكننا الحصول على الحد الأدنى لـ  $n-1$  مقارنة لمسألة تحديد أصغر عنصر. تخيل أن أي خوارزمية تحدد أصغر عنصر هي مباريات دُوري بين العناصر، وأن كل مقارنة هي مباراة في الدوري يربحها أصغر العنصرين. وبملاحظة أن كل عنصر ما عدا الرابع سيخسر على الأقل مباراة واحدة، يمكننا أن نستنتج أننا بحاجة إلى  $n-1$  مقارنة لتحديد العنصر الأصغر. وعلى ذلك فإن الخوارزمية MINIMUM أمثلية فيما يتعلق بعدد المقارنات المنجزة.

### إيجاد الأصغر والأكبر في آن واحد (مقا)

يتعرّن علينا، في بعض التطبيقات، إيجاد العنصر الأصغر والعنصر الأكبر لمجموعة من  $n$  عنصرًا في آنٍ معًا. فمثلًا، يمكن أن يتطلب برنامج بياني تقييس مجموعة من المعطيات  $(x, y)$ ، لتوافق مع شاشة إظهار مستطيلة أو تجهيزة إظهار بيانية أخرى. ولتحقيق ذلك، يجب أن يوجد البرنامج أولاً القيمة الصغرى والكبرى لكل إحداثي.

يتبقى، في هذه المرحلة، أن نتضح كيفية إيجاد الأصغر والأكبر معًا لـ  $n$  عنصرًا بإجراء  $\Theta(n)$  مقارنة، وهي أمثلة تقاربيًا: نوجد الأصغر والأكبر بصورة مستقلة باستخدام  $n-1$  مقارنة لكل منهما، أي  $2n-2$  مقارنة لكليهما.

في الحقيقة، يمكننا إيجاد الأصغر والأكبر معًا بإجراء  $3\lceil n/2 \rceil$  مقارنة على الأكثر. يمكننا فعل ذلك بالاحتفاظ بالأصغر والأكبر اللذين عُثِر عليهما آنفًا. وعوضًا عن معالجة كل عنصر دُخِل بمقارنته بالأصغر والأكبر الحاليين، بتكلفة مقارنتين لكل عنصر، نعالج العناصر أزواجًا. وذلك بأن نبدأ بمقارنة زوج من عناصر

المدخل  $i$  أحدهما بالأحمر، ثم نقارن أصغرها بالأصغر الحالي وأكبرها بالأكبر الحالي، وتكلفة هذا هو ثلاث مقارنات لكل عنصرين.

يعتمد تحديد القيمتين الابتدائيتين للأصغر والأكبر الحاليين على كون  $n$  عددًا فرديًا أو زوجيًا. فإذا كان  $n$  فرديًا، فإننا نجعل الأصغر والأكبر كليهما يساوي قيمة العنصر الأول، ثم نعالج العناصر المتبقية أزواجًا. وإذا كان  $n$  زوجيًا، فإننا ننجز مقارنة بين العنصرين الأولين لتحديد القيمتين الابتدائيتين للأصغر والأكبر، ثم نعالج العناصر المتبقية أزواجًا كما في حالة  $n$  الفردية.

لنحلّل الآن العدد الكلي للمقارنات: إذا كان  $n$  فرديًا، فإننا ننجز  $3\lfloor n/2 \rfloor$  مقارنة. وإذا كان  $n$  زوجيًا، فإننا ننجز عملية مقارنة أولية واحدة يتبعها  $3(n-2)/2$  مقارنة، أي  $3n/2 - 2$  مقارنة كلية. وهكذا، فإن العدد الكلي للمقارنات يساوي  $-$  في كلتا الحالتين  $3\lfloor n/2 \rfloor$  على الأكثر.

## تعاريف

### 1-1.9

يُزَيّن أنه يمكننا إيجاد العنصر الثاني في الصغر لـ  $n$  عنصراً بإجراء  $n + \lfloor \lg n \rfloor - 2$  مقارنة في أسوأ الحالات. (تلميح: أوجد أيضاً العنصر الأصغر.)

### 2-1.9

برهن أن  $3n/2 - 2$  هو الحد الأدنى لعدد عمليات المقارنة اللازم لإيجاد الأصغر والأكبر معاً لـ  $n$  عنصراً في أسوأ الحالات. (تلميح: ادرس عدد الأعداد التي يمكن أن تكون إما الأصغر وإما الأكبر، وتحرّر كيف تؤثر المقارنة على هذه الأعداد.)

## الاختيار بزم من خطي متوقع

### 2.9

تبدو مسألة الاختيار العامة أكثر صعوبة من المسألة البسيطة لإيجاد الأصغر. ومع ذلك، فإن من المدهش أن زمن التنفيذ المقارب لكلا المسألتين هو نفسه: وهو  $\Theta(n)$ . سنقدم في هذا المقطع خوارزمية فرق-تسد  $\text{divide-and-conquer}$  لحل مسألة الاختيار. تُعدّلت الخوارزمية  $\text{RANDOMIZED-SELECT}$  على نمط خوارزمية الفرز السريع المشروحة في الفصل 7. وكما في الفرز السريع، فإننا نعيّز صفيفة الدخول عوّديًا. ولكن على عكس الفرز السريع، الذي يعالج كلا طريقي التحزئة، فإن  $\text{RANDOMIZED-SELECT}$  يعمل على طرف واحد فقط من التحزئة. يتضح هذا الاختلاف في التحليل: ففي حين أن زمن التنفيذ المتوقع للفرز السريع هو  $\Theta(n \lg n)$ ، فإن زمن التنفيذ المتوقع لـ  $\text{RANDOMIZED-SELECT}$  هو  $\Theta(n)$ ، بافتراض أن العناصر متمايزة. يُستخدم الإجراء  $\text{RANDOMIZED-SELECT}$  الإجراء  $\text{RANDOMIZED-PARTITION}$  المتقدّم في المقطع 3.7.

ومن ثم، فإن هذا الإجراء، كما في RANDOMIZED-QUICKSORT، هو خوارزمية ذات عشوائية مضافة randomized algorithm، لتكون سلوكها يتحدد جزئيًا بواسطة خرج مولّد أعداد عشوائية. يعيد الرماز التالي لـ RANDOMIZED-SELECT العنصر الأصغر ذي الترتيب  $i$  للصفيفة  $A[p..r]$ .

```

RANDOMIZED-SELECT( $A, p, r, i$ )
1  if  $p == r$ 
2      return  $A[p]$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
4   $k = q - p + 1$ 
5  if  $i == k$  // the pivot value is the answer
6      return  $A[q]$ 
7  elseif  $i < k$ 
8      return RANDOMIZED-SELECT( $A, p, q - 1, i$ )
9  else return RANDOMIZED-SELECT( $A, q + 1, r, i - k$ )
    
```

يعمل الإجراء RANDOMIZED-PARTITION كما يلي. يفحص السطر 1 الحالة الأساسية للفؤدية، التي تتكون فيها الصفيفة الجزئية  $A[p..r]$  من عنصر واحد. في هذه الحالة، يجب أن تساوي  $i$  القيمة 1، ونعيد ببساطة  $A[p]$  في السطر الثاني على أنه العنصر الأصغر ذو الترتيب  $i$ . وإلا فإن استدعاء RANDOMIZED-PARTITION في السطر الثالث من الخوارزمية، يجرّئ الصفيفة  $A[p..r]$  إلى صفيقتين جزئيتين (يمكن أن تكونا خاليتين)  $A[p..q-1]$  و  $A[q+1..r]$  بحيث يكون كل عنصر من  $A[p..q-1]$  أصغر أو يساوي  $A[q]$ ، والذي بدوره أصغر من كل عنصر في  $A[q+1..r]$ . وكما في الفرز السريع، سوف نشير إلى  $A[q]$  بالعنصر **المحوري** *pivot*. يُحسب السطر 4 عدد العناصر  $k$  في الصفيفة الجزئية  $A[p..q]$ ، أي عدد العناصر في الجانب الأيمن من التجزئة زائد واحد هو العنصر المحوري. بعدها، يفحص السطر 5: هل  $A[q]$  هو العنصر الأصغر ذو الترتيب  $i$ ؟ فإذا كان الأمر كذلك، يعيد السطر 6 العنصر  $A[q]$ . وإلا فإن الخوارزمية تحدّد في أيّ من الصفيقتين الجزئيتين  $A[p..q-1]$  و  $A[q+1..r]$  يقع العنصر الأصغر ذو الترتيب  $i$ . فإذا كان  $i < k$ ، فإن العنصر المنشود يقع في الجانب الأيمن من التجزئة، ويختاره السطر 8 عُدديًا من الصفيفة الجزئية. وإذا كان  $i > k$ ، فإن العنصر المنشود يقع في الجانب الأعلى من التجزئة. ولما كنا نعلم سلفًا وجود  $k$  قيمةً جميعها أصغر من العنصر الأصغر ذي الترتيب  $i$  لـ  $A[p..r]$  (وهي بالتحديد عناصر الصفيفة  $A[p..q]$ ) فإن العنصر المنشود هو العنصر الأصغر ذو الترتيب  $(i - k)$  لـ  $A[q+1..r]$ ، وهو الذي يعثر عليه السطر 9 عُدديًا. يبدو أن الرماز يُسمح باستدعاء عُدديّ لصفيفة جزئية عالية العناصر. سيُطلب إليك في التمرين 2.9-1 أن تبين أن هذه الحالة غير ممكنة الحدوث.

إن زمن تنفيذ RANDOMIZED-SELECT في أسوأ الحالات هو  $\Theta(n^2)$ ، حتى في حال إيجاد الأصغر، لأنه من الممكن أن نكون سيّمي الحظ غامًا ونجرّئ دائمًا حول أكبر العناصر المثيقية، وتستغرق التجزئة

زمنًا  $\Theta(n)$ . وسوف نرى أن للخوارزمية زمن تنفيذ متوقع خطي، لذلك ولكونها ذات عشوائية مضافة، فلا يوجد أي دخل خاص يظهر السلوك في أسوأ الحالات.

لتحليل زمن التنفيذ المتوقع لـ RANDOMIZED-SELECT، سنفترض أن زمن التنفيذ للصفيفة  $A[p..r]$  من  $n$  عنصرًا هو متحول عشوائي نشير إليه بـ  $T(n)$ ، ونحصل على الحد الأعلى لـ  $E[T(n)]$  كما يلي. يعيد الإجراء RANDOMIZED-PARTITION باحتمالات متساوية أي عنصر على أنه العنصر المخوري. لذلك، فإن الصفيفة  $A[p..q]$  لها  $k$  عنصرًا (جميعها أصغر من العنصر المخوري أو تساويه) باحتمال  $1/n$ ، وذلك لكل  $k$  حيث  $1 \leq k \leq n$ . نُمَرِّف للقيم  $k = 1, 2, \dots, n$ ، منحولات عشوائية مؤشرة indicator random variables  $X_k$  حيث:

$X_k = 1$  {the subarray  $A[p..q]$  has exactly  $k$  elements} .

ومنه، إذا افترضنا أن العناصر متمايزة، يكون لدينا

$$E[X_k] = 1/n . \quad (1.9)$$

حين نستدعي RANDOMIZED-SELECT ونختار  $A[q]$  باعتباره عنصرًا محوريًا، فإننا لا نعلم سلفًا إذا كنا سنُنتهي فورًا بالمخواب الصحيح، أم سنستدعي الإجراء عُدديًا للصفيفة الجزئية  $A[p..q-1]$ ، أم سنستدعي الإجراء عُدديًا للصفيفة الجزئية  $A[q+1..r]$ . يعتمد هذا القرار على مكان وقوع العنصر الأصغر ذي الترتيب  $k$  بالنسبة إلى  $A[q]$ . فإذا افترضنا أن  $T(n)$  متزايد باطراد، أمكننا تقييد الزمن-الأعلى اللازم للاستدعاء العُددي بالزمن اللازم للاستدعاء العُددي لأكبر دخل ممكن. وبعبارة أخرى، للحصول على حد أعلى، سنفترض أن العنصر ذا الترتيب  $k$  يقع دائمًا في جانب الشحنة التي لها أكبر عدد من العناصر. إذا كان لدينا استدعاء معطى للإجراء RANDOMIZED-SELECT، فإن المنحول العشوائي المؤشر  $X_k$  يأخذ القيمة 1 لإحدى قيم  $k$  فقط، والقيمة 0 لقيم  $k$  الأخرى. فإذا كان  $X_k = 1$ ، فإن حجم العنصرين الجزئيتين (التي يمكن أن يستدعيهما الإجراء عُدديًا) هو:  $k-1$  و  $n-k$ . ويكون لدينا التكرار:

$$\begin{aligned} T(n) &\leq \sum_{k=1}^n X_k \cdot (T(\max(k-1, n-k)) + O(n)) \\ &= \sum_{k=1}^n X_k \cdot T(\max(k-1, n-k)) + O(n) . \end{aligned}$$

وبأخذ القيم المتوقعة للطرفين، يكون لدينا

$$\begin{aligned} E[T(n)] &\leq E \left[ \sum_{k=1}^n X_k \cdot T(\max(k-1, n-k)) + O(n) \right] \\ &= \sum_{k=1}^n E[X_k \cdot T(\max(k-1, n-k))] + O(n) \quad (\text{من خطية التوقع}) \end{aligned}$$

$$= \sum_{k=1}^n E[X_k] \cdot E[T(\max(k-1, n-k))] + O(n) \quad ((\text{من العلاقة (ت-24)})$$

$$= \sum_{k=1}^n \frac{1}{n} \cdot E[T(\max(k-1, n-k))] + O(n) . \quad ((\text{من العلاقة (1.9)})$$

ولتطبيق العلاقة (ت-24)، فإننا نُعَوِّل على كون  $X_k$  و  $T(\max(k-1, n-k))$  متحولين عشوائيين مستقلين. يُطلب إليك في التمرين 2-2.9 تبرير هذا الادعاء.

ليكن لدينا التعبير  $\max(k-1, n-k)$  لدينا

$$\max(k-1, n-k) = \begin{cases} k-1 & \text{if } k > \lfloor n/2 \rfloor , \\ n-k & \text{if } k \leq \lfloor n/2 \rfloor . \end{cases}$$

فإذا كان  $n$  زوجيًا، يظهر كل حدٍّ من الحدود من  $T(\lfloor n/2 \rfloor)$  إلى  $T(n-1)$  مرتين تمامًا في المجموع، وإذا كان  $n$  فرديًا، تظهر كل هذه الحدود مرتين ويظهر الحد  $T(\lfloor n/2 \rfloor)$  مرة واحدة. وبذلك يكون لدينا:

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + O(n) .$$

يتبيّن بالتعميوس أن  $E[T(n)] = O(n)$ . لنفترض أن  $E[T(n)] \leq cn$  ثابت  $c$  يحقق الشروط الابتدائية للعقودية. سنفترض أن  $T(n) = O(1)$  لقيم  $n$  التي هي أقل من ثابت ما (نحدّه لاحقًا). وسوف نختار أيضًا ثابتًا  $a$  بحيث تكون القيمة العظمى للدالة المُوصوفة بالحد  $O(n)$  المذكور آنفاً (والذي يصف المركبة غير العقودية لزمس تنفيذ الخوارزمية) محدودًا بالقيمة  $an$  لجميع قيم  $n > 0$ . وباستخدام هذه الفرضية الاستقرائية، يكون لدينا:

$$\begin{aligned} E[T(n)] &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an \\ &= \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\ &= \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an \\ &\leq \frac{2c}{n} \left( \frac{(n-1)n}{2} - \frac{(n/2 - 2)(n/2 - 1)}{2} \right) + an \\ &= \frac{2c}{n} \left( \frac{n^2 - n}{2} - \frac{n^2/4 - 3n/2 + 2}{2} \right) + an \\ &= \frac{c}{n} \left( \frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an \end{aligned}$$



$$\begin{aligned}
 &= c \left( \frac{3n}{4} + \frac{1}{2} - \frac{2}{n} \right) + an \\
 &\leq \frac{3cn}{4} + \frac{c}{2} + an \\
 &= cn - \left( \frac{cn}{4} - \frac{c}{2} - an \right).
 \end{aligned}$$

نحتاج، بغية استكمال البرهان، أن نبين أنه إذا كانت قيمة  $n$  كبيرة بقدر كاف، فإن التعبير الأخير يساوي على الأكثر  $cn$  أو، على نحو مكافئ، أن  $cn/4 - c/2 - an \geq 0$ ، فإذا أضفنا  $c/2$  إلى كلا الطرفين وأخرجنا العامل المشترك  $n$ ، نحصل على  $n(c/4 - a) \geq c/2$ . فإذا اخترنا الثابت  $c$  بحيث  $c/4 - a > 0$ ، أي  $c > 4a$ ، أمكننا تقسيم كلا الطرفين على  $c/4 - a$ ، ونحصل على:

$$n \geq \frac{c/2}{c/4 - a} = \frac{2c}{c - 4a}.$$

وهكذا، إذا افترضنا أن  $T(n) = O(1)$  لقيم  $n$  التي تحقق  $n < 2c/(c - 4a)$ . فإن  $E[T(n)] = O(n)$ . نستنتج أنه يمكننا إيجاد أيّ إحصائية ترتيب، وعصوفاً الوسط، بزمن خطي متوقع إذا افترضنا أن العناصر متمايزة.

## لمارين

### 1-2.9

بين أن RANDOMIZED-SELECT لا تُحدث أبداً استدعاءً عودياً لصفيفة من 0 عنصرًا.

### 2-2.9

أثبت أن المنحول العشوائي المؤشر  $X_k$  والقيمة  $T(\max(k-1, n-k))$  مستقلان أحدهما عن الآخر.

### 3-2.9

اكتب نسخة تكرارية لـ RANDOMIZED-SELECT.

### 4-2.9

افترض أننا نستخدم RANDOMIZED-SELECT لاختيار العنصر الأصغر للصفيفة

$A = \langle 3, 2, 9, 0, 7, 5, 4, 8, 6, 1 \rangle$ . صف متاليةً من التجزئات التي تعطي أسوأ أداء

لـ RANDOMIZED-SELECT.

## 3.9 الاختيار بزمن خطي في أسوأ الحالات

سندرس الآن خوارزمية اختيار زمن تنفيذها  $O(n)$  في أسوأ الحالات. إن الخوارزمية SELECT، كما هو الحال في RANDOMIZED-SELECT، تُجد العنصر لطلوب بتجزئة صفيفة الدحل عودياً. غير أننا نضمن هنا تفريقاً

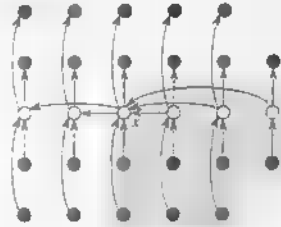
جيداً للصفيفة أثناء تجزئتها. تستخدم SELECT خوارزمية التجزئة الاحتمية PARTITION المستخدمة في الفرز السريع (انظر المقطع 1.7)، ولكنها مُعدّلة لتأخذ العنصر الذي تجري التجزئة حوله باعتباره متوسط دخل. تحدد الخوارزمية SELECT العنصر الأصغر ذا الترتيب  $i$  لصفيفة دخل من  $n > 1$  عنصراً متمايزاً بتنفيذ الخطوات التالية. (إذا كان  $n = 1$ ، فإن SELECT تعيد ببساطة قيمة الدخل الوحيد بوصفه العنصر الأصغر ذا الترتيب 1.)

1. قسّم الـ  $n$  عنصراً لصفيفة الدخل إلى  $\lceil n/5 \rceil$  مجموعة كل منها من 5 عناصر ومجموعة واحدة على الأكثر تُشعبها من العناصر المتبقية من قسمة  $n$  على 5.
2. اكتشف الوسط لكل من المجموعات الـ  $\lceil n/5 \rceil$  باستخدام فرز بالإدراج لفرز عناصر كل مجموعة (لكل منها 5 عناصر على الأكثر) ثم اختر الوسط من اللائحة المفروزة لعناصر المجموعة.
3. استخدم SELECT عُددياً لاكتشاف الوسط  $x$  من الأوساط الـ  $\lceil n/5 \rceil$  التي اكتشفتها في الخطوة 2. (إذا كان عدد الأوساط زوجياً، فإن  $x$  بحسب اصطلاحنا هو الوسط الأدنى.)
4. تجزئ صفيفة الدخل حول وسط الأوساط  $x$  باستخدام نسخة مُعدّلة من PARTITION. لتكن  $k$  أكبر بواحد من عدد العناصر في الجانب الأدنى من التجزئة، أي إن  $x$  هو العنصر الأصغر ذو الترتيب  $k$  ويوجد  $n - k - 1$  عنصراً في الجانب الأعلى من التجزئة.
5. إذا كان  $k = i$ ، فأعدّ  $x$ . وإذا كان  $k < i$ ، فاستخدم SELECT عُددياً لاكتشاف العنصر الأصغر ذي الترتيب  $i$  في الجانب الأدنى. وإذا كان  $k > i$ ، فاستخدم SELECT عُددياً لاكتشاف العنصر الأصغر ذي الترتيب  $k - i$  في الجانب الأعلى.

لتحليل زمن تنفيذ SELECT، نحدّد بدايةً الحد الأدنى لعدد العناصر التي هي أكبر من عنصر التجزئة  $x$ . يساعدنا الشكل 1.9 في إظهار هذه الخطوات. إن نصف الأوساط - على الأقل - التي اكتشفناها في الخطوة 2 أكبر من وسط الأوساط  $x$  أو تساويه<sup>1</sup>. ومنه، فإن نصف المجموعات الـ  $\lceil n/5 \rceil$  على الأقل تتضمن 3 عناصر على الأقل أكبر من  $x$ ، ما عدا المجموعتين التاليتين: المجموعة التي عناصرها أقل من 5 عناصر في حال كانت  $n$  لا تقبل القسمة تماماً على 5، والمجموعة التي تحتوي  $x$  نفسه. وباستبعاد هاتين المجموعتين، فإن عدد العناصر التي هي أكبر من  $x$  يساوي على الأقل

$$3 \left( \left\lceil \frac{1}{2} \left\lceil \frac{n}{5} \right\rceil \right\rceil - 2 \right) \geq \frac{3n}{10} - 6 .$$

<sup>1</sup> بسبب افتراضنا أن العناصر متمايزة، فإن جميع الأوساط - ما عدا  $x$  - هي إما أكبر من  $x$  وإما أصغر من  $x$ .



**الشكل 1.9** تحليل الخوارزمية SELECT. جرى تمثيل العناصر الـ  $n$  بدوائر صغيرة، وكل مجموعة من خمس عناصر تشغل عموداً. لُكِّت أوساط المجموعات بالأبيض، ووُضِعَت لصاقة إلى جانب وسط الأوساط  $x$ . (عندما يكون الوسط لعدد زوجي من العناصر، فإننا نستخدم الوسط الأدنى.) تتجه الأسهم من العناصر الكبرى إلى العناصر الصغرى، والتي منها يمكن ملاحظة أن 3 من كل مجموعة مثثلة بـ 5 عناصر إلى يمين  $x$  هي أكبر من  $x$ ، وأن 3 من كل مجموعة من 5 عناصر إلى يسار  $x$  هي أقل من  $x$ . العناصر التي غُلِبَتْ أقلها أكبر من  $x$  على نظير خلفية مظلمة.

وبالمثل، فإن  $3n/10 - 6$  عنصراً على الأقل هي أقل من  $x$ . وهكذا، في أسوأ الحالات، فإن الخطوة 5 ستدعي SELECT غُدِّباً لـ  $7n/10 + 6$  عنصراً على الأكثر.

يمكننا الآن استنتاج علاقة عُدَّدية لزمن التنفيذ في أسوأ الحالات  $T(n)$  للخوارزمية SELECT. تستغرق الخطوات 1 و 2 و 4 زمناً  $O(n)$ . (تتضمن الخطوة 2 استدعاءً لفرز بالإدراج على مجموعات أحجامها  $O(1)$ .) تستغرق الخطوة 3 زمناً  $T(\lceil n/5 \rceil)$ ، وتستغرق الخطوة 5 على الأكثر زمناً  $T(7n/10 + 6)$ ، بافتراض أن  $T$  متزايدة باطراد. سنفترض فرضية، تبدو في البداية غير مبررة، وهي أن أيّ دخل أقل من 140 عنصراً يحتاج إلى زمن  $O(1)$ ؛ سنوضح قريباً منشأ هذا الثابت السحري 140. لذا يمكننا الحصول على العُدَّدية

$$T(n) \leq \begin{cases} O(1) & \text{if } n < 140 \\ T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) & \text{if } n \geq 140 \end{cases}$$

سنبين أن زمن التنفيذ خططي بالتعويض. وبعبارة أدق، سنبين أن  $T(n) \leq cn$  لقيم الثابت  $c$  الكبيرة بقدر مناسب ولجميع قيم  $n > 0$ . سنبدأ بافتراض أن  $T(n) \leq cn$  لقيم الثابت  $c$  الكبيرة بقدر مناسب ولجميع قيم  $n < 140$ ؛ وهذا الفرضية مخفَّعة إذا كان  $c$  كبيراً بقدر كافٍ. سوف نحدد أيضاً الثابت  $a$  بحيث تكون قيمة الدالة الموصوفة بالحد  $O(n)$  المذكور آنفاً (والذي يَصفُ المكوّن غير العُدَّدي لزمن تنفيذ الخوارزمية) محدوداً بالقيمة  $an$  لجميع قيم  $n > 0$ . بتعويض هذا الفرضية الاستقرائية في الطرف الأيمن للعُدَّدية نحصل على:

$$\begin{aligned} T(n) &\leq c\lceil n/5 \rceil + c(7n/10 + 6) + an \\ &\leq cn/5 + c + 7cn/10 + 6c + an \end{aligned}$$

$$= 9cn/10 + 7c + an$$

$$= cn + (-cn/10 + 7c + an) ,$$

الذي يساوي على الأكثر  $cn$  إذا كان

$$-cn/10 + 7c + an \leq 0 . \quad (2.9)$$

إن المتراجحة (2.9) مكافئة للمتراجحة  $(c \geq 10a(n/(n-70)))$  في حال  $n > 70$ . وحيث أننا افترضنا أن  $n \geq 140$ ، يكون لدينا  $n/(n-70) \leq 2$ ، وسيحقق اختبارنا  $c \geq 20a$  المتراجحة (2.9). (لاحظ أنه لا يوجد شيء خاص بالثابت 140؛ فبإمكاننا الاستعاضة عنه بأي عدد صحيح أكبر تمامًا من 70 ثم نختار  $c$  وفقًا له.) وعلى ذلك، فإن زمن تنفيذ SELECT في أسوأ الحالات خطي.

إن الإجراءين SELECT و RANDOMIZED-SELECT يحددان، كما في الفرز بالمقارنة (انظر المقطع 1.8)، معلومات عن الترتيب النسبي للعناصر بمقارنة العناصر فقط. نذكر (من الفصل 8) أن الفرز يحتاج زمنًا  $\Omega(n \lg n)$  في نموذج المقارنة، وحتى وسيطًا (انظر المسألة 1-8). تضع خوارزميات الفرز بالزمن-الخطي في الفصل 8 افتراضات عن الدخول. وبالمقابل، لا تحتاج خوارزميات الاختيار بالزمن-الخطي في هذا الفصل إلى أي افتراضات عن الدخول. وهي لا تخضع للحد الأدنى  $\Omega(n \lg n)$ ، لكونها قادرة على حل مسألة الاختيار من دون فرز. وهكذا، فإن حل مسألة الاختيار باستخدام طريقة الفرز والفهرسة (sorting and indexing)، كما عُرضت في مقدمة هذا الفصل، هو حل غير فعالٍ على نحو مقارب.

## تمارين

### 1-3.9

في خوارزمية SELECT، نقسم عناصر الدخول إلى مجموعات كلٍّ منها من 5 عناصر. هل تعمل الخوارزمية بزمن خطي لو قسمنا عناصر الدخول إلى مجموعات كلٍّ منها من 7 عناصر؟ ناقش أن SELECT لا تُنفذ بزمن خطي إذا استخدمنا مجموعات كلٍّ منها من 3 عناصر.

### 2-3.9

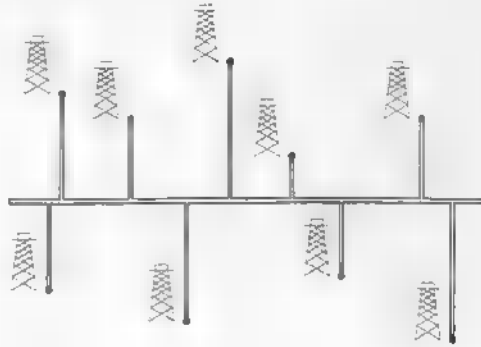
حلل الإجراء SELECT لتبين أنه في حال  $n \geq 140$ ، فإن  $\lceil n/4 \rceil$  عنصرًا على الأقل أكبر من وسط الأوساط  $x$ ، وأن  $\lceil n/4 \rceil$  عنصرًا على الأقل أقل من  $x$ .

### 3-3.9

بين كيف يمكن جعل الفرز السريع يُنفذ بزمن  $O(n \lg n)$  في أسوأ الحالات، افترض أن جميع العناصر متمايزة.

### \* 4-3.9

افترض أن خوارزمية تستخدم مقارناتٍ فقط لاكتشاف العنصر الأصغر ذي الترتيب  $i$  في مجموعة من  $n$  عنصرًا. بين أنه يمكن لهذه الخوارزمية أيضًا اكتشاف العنصر الأصغر ذي الترتيب  $i-1$  والعنصر الأكبر ذي الترتيب  $i-n$  دون إجراء أي عملياتٍ مقارنةٍ إضافية.



**الشكل 2.9** يحتاج السيد أولي إلى تحديد موقع خط أنابيب نفط يتجه من الشرق إلى الغرب يجعل الطول الكلي للوصلات الفرعية المشعبة شمالاً أو جنوباً أصغر.

### 5-3.9

افترض أن لديك مسألاً فرعياً هو "صندوق-أسود" black-box لاكتشاف الوسط بزم حطلي في أسوأ الحالات. أعط خوارزمية بسيطة تحل مسألة الاختيار لإحصائية ترتيب اعتباطية بزم-حطلي.

### 6-3.9

إن الكميات ذات الترتيب  $k$  ( $k$ th quantiles) مجموعة من  $n$  عنصراً هي إحصائيات الترتيب  $k - 1$  التي تقسم مجموعة مفروزة إلى مجموعات متساوية-الحجم (باختلاف 1 على الأكثر). أعط خوارزمية تسرد الكميات ذات الترتيب  $k$  لمجموعة بزم  $O(n \lg k)$ .

### 7-3.9

حيث خوارزمية تُنفذ بزم  $O(n)$  وتحدد مجموعة معطاة  $S$  من  $n$  عنصراً متميزاً وعدد صحيح موجب  $k$  ( $k \leq n$ ) إلى  $k$  عدداً من  $S$  التي هي أقرب ما يكون إلى وسط المجموعة  $S$ .

### 8-3.9

لتكن  $X[1..n]$  و  $Y[1..n]$  صفيفان، تتضمن كل منهما  $n$  عنصراً بترتيب مفروز سلفاً. أعط خوارزمية، تُنفذ بزم  $O(\lg n)$ ، تكتشف وسط جميع عناصر  $2n$  الصفيفين  $X$  و  $Y$ .

### 9-3.9

السيد أولي Olay مستشار لشركة نفط تخطط لمد خط أنابيب ضخيم يتجه من الشرق إلى الغرب عبر حقل نفط فيه  $n$  بئر. ترغب الشركة في وصل خط أنابيب فرعياً مباشراً من كل بئر إلى خط الأنابيب الرئيس عبر الطريق الأقصر (شمالاً أو جنوباً)، كما هو مبين في الشكل 2.9. إذا أُعطينا الإحداثيات  $x$  و  $y$  للآبار، كيف

يمكن للسيد أولي اختيار الموقع الأمثل لخط الأنابيب الرئيس، الذي يجعل الطول الكلي للوصلات الفرعية أصغر؟ بَيِّن كيف يمكن تحديد الموقع الأمثل بزمَن خطي.

## مسائل

### 1-9 أكبر عددًا في ترتيب مفروز

لنكن لدينا مجموعة من  $n$  عددًا، ونرغب في اكتشاف أكبر  $i$  عددًا في ترتيب مفروز largest  $i$  numbers in sorted order باستخدام خوارزمية تعتمد على المقارنات. أوجد خوارزمية تنجز كلاً من الطرق الآتية بأفضل زمن تنفيذ مقارب في أسوأ الحالات، وحلّل زمن تنفيذ الخوارزمية بدلالة  $n$  و  $i$ .

أ. فرز الأعداد واسرد أكبر  $i$  عددًا.

ب. ابنِ رتلاً ذا أولوية الأكبر max-priority queue من الأعداد، واستدع الإجراء EXTRACT-MAX  $i$  مرةً.

ت. استخدم خوارزمية إحصائية-ترتيب لاكتشاف العدد ذي الترتيب  $i$  في الكبر، جرّئ حول هذا العدد، وافرز أكبر  $i$  عددًا.

### 2-9 الوسط المثقل

إن الوسط (الأدنى) المثقل (lower) weighted median لـ  $n$  عنصرًا متساويًا  $x_1, x_2, \dots, x_n$  ذات أوزان موجبة  $\omega_1, \omega_2, \dots, \omega_n$  تحقق  $\sum_{i=1}^n \omega_i = 1$ ، هو العنصر  $x_k$  الذي يحقق:

$$\sum_{x_i < x_k} \omega_i < \frac{1}{2}$$

و

$$\sum_{x_i > x_k} \omega_i \leq \frac{1}{2}.$$

فمثلاً، إذا كانت العناصر تساوي 0.1, 0.35, 0.05, 0.1, 0.15, 0.05, 0.2، وكل عنصر يساوي وزنه (أي،  $\omega_i = x_i$  لكل  $i = 1, 2, \dots, 7$ )، فإن الوسط يساوي 0.1، في حين أن الوسط المثقل يساوي 0.2.

أ. برهن أن وسط  $x_1, x_2, \dots, x_n$  هو الوسط المثقل لـ  $x_i$  بالأوزان  $\omega_i = 1/n$  لكل  $i = 1, 2, \dots, n$ .

ب. بَيِّن كيف نحسب الوسط المثقل لـ  $n$  عنصرًا بزمَن  $O(n \lg n)$  في أسوأ الحالات باستخدام الفرز.

ت. بَيِّن كيف نحسب الوسط المثقل بزمَن  $\Theta(n)$  في أسوأ الحالات باستخدام خوارزمية الوسط بزمَن خطي

linear-time median algorithm كالإجراء SELECT المذكور في لقطع 3.9.

نعرّف مسألة تحديد موقع مكتب البريد *post-office location problem* كما يلي. ليكن لدينا  $n$  نقطة  $p_1, p_2, \dots, p_n$  ترتبط بها الأوزان  $\omega_1, \omega_2, \dots, \omega_n$ . نرغب بإيجاد النقطة  $p$  (ليست بالضرورة أن تكون إحدى نقاط الدخل) التي تجعل المجموع  $\sum_{i=1}^n \omega_i d(p, p_i)$  أصغرًا، حيث  $d(a, b)$  المسافة بين النقطتين  $a$  و  $b$ .

ث. برهن أن الوسط المتكافئ هو الحل الأفضل لمسألة تحديد موقع مكتب بريد أحادي البعد 1-dimensional، بافتراض أن النقاط أعدادًا حقيقية، والبعد بين النقطتين  $a$  و  $b$  يساوي  $d(a, b) = |a - b|$ .

ج. أوجد أفضل حل لمسألة تحديد موقع مكتب بريد ثنائي البعد، بافتراض أن النقاط هي أزواج الإحداثيات  $(x, y)$ ، والمسافة بين النقطتين  $\blacksquare = (x_1, y_1)$  و  $b = (x_2, y_2)$  هي مسافة مانهاتن *Manhattan distance* التي تُعطى بالعلاقة  $d(a, b) = |x_1 - x_2| + |y_1 - y_2|$ .

### 3-9 إحصائيات ترتيب صغير

يُتَبادَلُ سابقًا أن عدد المقارنات في أسوأ الحالات  $T(n)$  الذي يستخدمه الإجراء SELECT لاختيار الإحصائية ذات الترتيب  $i$  لـ  $n$  عددًا تحقق  $T(n) = \Theta(n)$ ، ولكن الثابت الخفي ضمن تدوين  $\Theta$  كبير جدًا. فإذا كان  $i$  صغيرًا بالنسبة إلى  $n$ ، يمكننا تجنب إجراء مختلف يستخدم SELECT باعتباره مساهمًا فرعيًا ولكنه يُجري مقارنات أقل في أسوأ الحالات.

أ. صف خوارزمية تستخدم  $U_i(n)$  مقارنة لاكتشاف العنصر الأصغر ذي الترتيب  $i$  لـ  $n$  عنصرًا، حيث:

$$U_i(n) = \begin{cases} T(n) & \text{if } i \geq n/2 \\ |n/2| + U_i(|n/2|) + T(2i) & \text{otherwise} \end{cases}$$

(نلاحظ: ابدأ بـ  $|n/2|$  مقارنة من الأزواج المنفصلة، وارجع عودًا إلى المجموعة التي تحتوي على أصغر عنصر من كل زوج.)

ب. يَبَيَّنُ أنه إذا كان  $i < n/2$  فإن  $U_i(n) = n + O(T(2i) \lg(n/i))$ .

ت. يَبَيَّنُ أنه إذا كان  $i$  ثابتًا أصغر من  $n/2$  فإن  $U_i(n) = n + O(\lg n)$ .

ث. يَبَيَّنُ أنه إذا كان  $i = n/k$  لكل  $k \geq 2$  فإن  $U_i(n) = n + O(T(2n/k) \lg k)$ .

### 4-9 تحليل بدلي (آخر) لمسألة اختيار ذات عشوائية مضافة

سنستخدم في هذه المسألة متحولات عشوائية مؤشّرة لتحليل الإجراء RANDOMIZED-SELECT على نحو مماثل لتحليلنا لـ RANDOMIZED-QUICKSORT المذكور في المقطع 2.4.7.

منفترض، كما في حالة تحليل الفرز السريع، أن جميع العناصر متمايزة، ونعيد تسمية عناصر صفيفة

الدخل  $A$  كما يلي:  $z_1, z_2, \dots, z_n$ ، حيث  $z_i$  هو العنصر الأصغر ذو الترتيب  $i$ . وبذلك، بعيد الاستدعاء RANDOMIZED-SELECT( $A, 1, n, k$ ) العنصر  $z_k$ .

ليكن:

$X_{ijk} = I\{z_k \text{ is compared with } z_j \text{ sometime during the execution of the algorithm to find } z_i\}$ .

لكل  $1 \leq i < j \leq n$

أ. أعط تعبيراً دقيقاً لـ  $E[X_{ijk}]$ . (تلميح: يمكن أن يأخذ تعبيرك قيماً مختلفة، تبعاً لقيم  $i$  و  $j$  و  $k$ ).

ب. لنرمز بـ  $X_k$  لعدد عمليات المقارنة الكلية بين عناصر الصيغة  $A$  المُنفذة خلال عملية اكتشاف  $z_k$ .  
بيّن أن:

$$E[X_k] \leq 2 \left( \sum_{i=1}^k \sum_{j=k}^n \frac{1}{j-i+1} + \sum_{j=k+1}^n \frac{j-k-1}{j-k+1} + \sum_{i=1}^{k-2} \frac{k-i-1}{k-i+1} \right).$$

ت. بيّن أن  $E[X_k] \leq 4$ .

ث. بافتراض أن جميع عناصر الصيغة  $A$  متميزة، استنتج أن RANDOMIZED-SELECT تُنفذ بـ زمن متوقع  $O(n)$ .

## ملاحظات الفصل

ابتكر Blum و Floyd و Pratt و Rivest و Tarjan [50] خوارزمية لاكتشاف الوسط بـ زمن خطي في أسوأ الحالات. وتعود أسرع نسخة ذات عشوائية مضافة إلى Hoare [169]. وكان Floyd و Rivest [108] قد طوروا نسخة مُحسّنة ذات عشوائية مضافة بحيث يجري اختيار التجزئات حول عنصر ما عَوْدُها من عينة صغيرة من العناصر.

ما زال عدد المقارنات اللازمة لتحديد الوسط غير معروف بالضبط. غير أن Bent و John [41] قدّما حدّاً أدنى يساوي  $2n$  مقارنة لاكتشاف الوسط، وقدّم Schönhage و Paterson و Pippenger [302] حدّاً أعلى يساوي  $3n$ . وحسّن Dor و Zwick هذين الحدّين. وكان الحدّ الأعلى الذي قدّمناه [94] أقلّ بقليل من  $2.95n$ ، والحدّ الأدنى [95] يساوي  $n(2 + \epsilon)$ ، حيث  $\epsilon$  عدد ثابت موجب صغير، وبذلك حسّنا قليلاً نتائج العمل الذي قام به Dor وآخرون [93]. وصنّف Paterson [272] بعض هذه النتائج إضافة إلى أعمال أخرى ذات صلة بها.





## تمهيد

تُعَدُّ المجموعات *sets* من الأساسيات في علم الحاسوب مثلما هو شأنها في الرياضيات. وفي حين أن المجموعات في الرياضيات لا تتغير، فإن المجموعات التي تتعامل معها الخوارزميات يمكن أن تكبر أو تصغر أو تتغير مع الزمن، لذا فإننا نَصِفُ مثل هذه المجموعات بأنها *ديناميكية dynamic*. تعرض الفصول الخمسة التالية بعض التقنيات الأساسية لتمثيل المجموعات الديناميكية المنتهية وكيفية التعامل معها حاسوبياً.

قد تطلب الخوارزميات تنفيذ عدة أنواع من العمليات على المجموعات. على سبيل المثال، يحتاج العديد من الخوارزميات إلى إمكانية إدراج عناصر في مجموعة، وحذف عناصر منها، واختبار الانتماء إليها. تسمى المجموعة الديناميكية التي تدعم هذه العمليات *معجماً dictionary*. وتطلب بعض الخوارزميات الأخرى عمليات أكثر تعقيداً. فمثلاً، ندعم أرنال ذات أولوية الأصغر أولاً - التي قدمناها في الفصل السادس ضمن سياق بنية المعطيات الكومة - عمليات إدراج عنصر في مجموعة، واستخراج العنصر الأصغر في مجموعة. وتعتمد الطريقة الفضلى لتنفيذ مجموعة ديناميكية على العمليات التي يجب أن تقدمها.

### عناصر المجموعة الديناميكية

في التنجيز النموذجي للمجموعة الديناميكية، يمثل كل عنصر بفرض، وفي حال وجود مؤشر يشير إلى هذا الفرض، يمكن فحص واصفاته والتعامل معها. (يناقش المقطع 3.10 تنجيز الأغراض والمؤشرات في بيانات البرمجة التي لا تحتوي هذه الأنماط باعتبارها أنماط معطيات أساسية.) وتفترض بعض أنواع المجموعات الديناميكية أن أحد واصفات الفرض هو *مفتاح key* معرف للفرض. إذا كانت جميع المفاتيح مختلفة، يمكننا عندها أن نعتبر المجموعة الديناميكية مجموعة قيم مفتاحية. وقد يجوي الفرض *معطيات تابعة satellite data* متضمنة في واصفات الفرض الأخرى، لكنها مع ذلك لا تُستخدم في تنجيز المجموعة. وقد تتضمن الأغراض كذلك واصفات يجري التعامل معها ضمن عمليات المجموعة، ويمكن أن تحتوي هذه الوصفات على معطيات

أو مؤشرات لأغراض أخرى في المجموعة.

نفترض بعض المجموعات الديناميكية سلفاً أن المفاتيح تنتمي إلى مجموعة مرتبة ترتيباً كلياً كمجموعة الأعداد الحقيقية أو مجموعة الكلمات المرتبة وفق الترتيب الأبجدي للمألوف. يتيح لنا الترتيب الكلي تعريف العنصر الأصغر في المجموعة مثلاً، أو الحديث عن العنصر التالي الأكبر من عنصر محدد في المجموعة.

### العمليات على المجموعات الديناميكية

يمكن تصنيف العمليات على المجموعات الديناميكية في صنفين: **الاستفسارات queries** التي نعيد ببساطة معلومات تتعلق بالمجموعة، و**عمليات التعديل modifying operations** التي نغير المجموعة. وفيما يلي قائمة العمليات النموذجية على المجموعات. يحتاج أي تطبيق محدد عادة إلى تنحيز بضع عمليات منها فقط.

#### SEARCH( $S, k$ )

استفسار يأخذ مجموعة معطاة  $S$  وقيمة المفتاح  $k$ ، ويعيد مؤشر  $x$  إلى عنصر في  $S$  يحقق  $x.key = k$ ، أو يعيد NIL إن لم يوجد في  $S$  مثل هذا العنصر.

#### INSERT( $S, x$ )

عملية تعديل تضيف العنصر المشار إليه بـ  $x$  إلى المجموعة  $S$ . نفترض عادة أنه قد سبق إعطاء قيمة ابتدائية لجميع واصفات العنصر ■ التي نحتاجها لتحيز المجموعة.

#### DELETE( $S, x$ )

عملية تعديل، تُزيل عنصر المجموعة المشار إليه بمؤشر  $x$  من المجموعة  $S$  (لاحظ أن هذه العملية تستخدم مؤشراً إلى عنصر  $x$  وليس قيمة مفتاح).

#### MINIMUM( $S$ )

استفسار يطبق على المجموعة  $S$  المرتبة ترتيباً كلياً، ويعيد مؤشراً إلى العنصر الذي يمتلك المفتاح الأصغر في  $S$ .

#### MAXIMUM( $S$ )

استفسار يطبق على المجموعة  $S$  المرتبة ترتيباً كلياً، ويعيد مؤشراً إلى العنصر الذي يمتلك المفتاح الأكبر في  $S$ .

#### SUCCESSOR( $S, x$ )

استفسار يأخذ عنصراً  $x$  قيمة مفتاحه موجودة في المجموعة  $S$  المرتبة ترتيباً كلياً، ويعيد مؤشراً إلى العنصر الأكبر منه مباشرة، أو يعيد NIL إذا كان  $x$  هو العنصر الأكبر في المجموعة.

#### PREDECESSOR( $S, x$ )

استفسار يأخذ عنصراً  $x$  قيمة مفتاحه موجودة في المجموعة  $S$  المرتبة ترتيباً كلياً، ويعيد مؤشراً إلى العنصر الأصغر منه مباشرة، أو يعيد NIL إذا كان  $x$  هو العنصر الأصغر في المجموعة.

في بعض الحالات، نستطيع توسيع الاستفسارتين SUCCESSOR و PREDECESSOR بحيث يمكن تطبيقهما على المجموعات التي تحوي عناصر غير متمايزة. ففي مجموعة تتألف من  $n$  مفتاحاً، من الطبيعي أن نفترض سلفاً أن استدعاء MINIMUM متبوعاً به  $n-1$  استدعاء لـ SUCCESSOR يعدد عناصر المجموعة بالترتيب.

نقيس الزمن الذي يستغرقه تنفيذ عملية من عمليات المجموعة عادةً بدلالة حجم المجموعة. فمثلاً، يصف الفصل 13 بنية معطيات تدعم جميع العمليات المشار إليها آنفاً على مجموعة حجمها  $n$  بزمن  $O(\lg n)$ .

### لمحة إلى الباب III

تصف الفصول من 10 إلى 14 عدة بنى معطيات يمكننا أن نستخدمها في تنجيز المجموعات الديناميكية. ونستخدم العديد من هذه البنى لاحقاً لبناء خوارزميات فعالة في مسائل مختلفة، وقد عرضنا في الفصل السادس بنية معطيات هامة أخرى هي الكومة.

يقدم الفصل 10 أساسيات التعامل مع بنى المعطيات البسيطة كالمكئوس، والرتل، والقائمة المترابطة، والشجرة ذات الجذر. ويبيّن كذلك كيف تنحصر الأغراض والمؤشرات في بيئات البرمجة التي لا تتضمن هذه البنى باعتبارها بنى أولية. فإذا سبى أن درّست مقررًا في مقدمات البرمجة، فستجد أن محتويات هذا الفصل مألوفة لديك.

يُعرّف الفصل 11 جداول التلييد hash tables التي تدعم العمليات للمجمعة INSERT و DELETE و SEARCH. يحتاج التلييد في أسوأ الحالات إلى زمن  $\Theta(n)$  لإنجاز عملية SEARCH، غير أن الزمن المتوقع للعمليات الخاصة بمجدول التلييد هو  $O(1)$ . يستند تحليل التلييد إلى الاحتمالات، غير أن غالبية الفصل لا تحتاج إلى خلفية عن هذا الموضوع.

تدعم أشجار البحث الثنائي التي يتناولها الفصل 12 جميع العمليات المتعلقة بالمجموعات الديناميكية الواردة آنفاً. وفي أسوأ الحالات تستغرق كل عملية زمناً  $\Theta(n)$  في حالة شجرة ذات  $n$  عنصرًا، ولكن في حالة شجرة بحث ثنائي مبنية بناء عشوائيًا، يقدر زمن كل عملية بـ  $O(\lg n)$ . وتعتبر أشجار البحث الثنائي أساساً للعديد من بنى المعطيات الأخرى.

يُعرّف الفصل 13 الأشجار الحمراء-السوداء، وهي شكل مختلف من أشكال أشجار البحث الثنائي. وعلى العكس من أشجار البحث الثنائي العادية، تضمن الأشجار الحمراء-السوداء أدائها الجيد، فعليًا تستغرق زمناً  $O(\lg n)$  في أسوأ الحالات. والشجرة الحمراء-السوداء هي شجرة بحث متوازنة. يقدم الفصل 18 في الباب 7 نوعاً آخر من الأشجار الثنائية للترابطة يدعى B-tree (الأشجار للمعممة). ومع أن آليات الأشجار الحمراء-السوداء معقدة نوعاً ما، إلا أنك تستطيع أن تتف على معظم خصائصها من الفصل دون دراسة آلياتها بالتفصيل. في جميع الأحوال، لا بد أنك ستجد أن التنقل ضمن الرماز مفيد جداً.

في الفصل 14، نبيّن كيف نُغني الأشجار الحمراء-السوداء لدعم عمليات أخرى غير تلك العمليات الأساسية الواردة آنفاً. وسنُغنيها أولاً بحيث نتمكن من دعم إحصائيات الترتيب ضمن مجموعة من المفاتيح، ثم نُغنيها بطريقة أخرى لدعم مجالات الأعداد الحقيقية.

## 10 بنى المعطيات الأولية

نبحث في هذا الفصل تمثيل المجموعات الديناميكية بنى معطيات بسيطة تُستخدم للمؤشرات. ومع أننا نستطيع بناء العديد من بنى المعطيات المعقدة باستخدام المؤشرات، إلا أننا نقدم هنا البنى الأولية فقط وهي: المكذّسات، والأرتال، واللوائح المترابطة، والأشجار ذات الجذر. نبيّن كذلك طرائق تركيب الأغراض والمؤشرات انطلاقاً من الصقيقات.

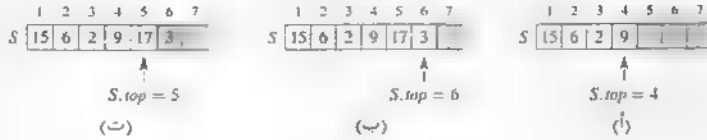
### 1.10 المكذّسات والأرتال

المكذّسات والأرتال مجموعات ديناميكية، تتميز بأن العنصر الذي يُحذف من المجموعة عند استخدام العملية DELETE عنصرٌ معروفٌ سلفاً. ففي المكثس *stack* يُحذف من المجموعة آخر عنصر أُدرج فيها، لذلك يقال بأن المكثس ينخّز استراتيجيّة الداخل آخرًا خارج أولًا *last-in, first-out* أو *LIFO*. وبالمثل، يُحذف دومًا من الرتل *queue* العنصر الذي قضى أطول زمن فيه؛ أي إن الرتل ينخّز استراتيجيّة الداخل أولًا خارج أولًا *first-in, first-out* أو *FIFO*. توجد عدة طرق فعّالة لتنخيز المكذّسات والأرتال ضمن الحاسوب. نبيّن في هذا المقطع كيف نستخدم صيغةً بسيطةً لتنخيز كلٍّ من هاتين البنىتين.

#### المكذّسات

تسمى العملية INSERT في المكذّسات غالبًا *PUSH*، وتسمى العملية DELETE التي لا تأخذ أيّ متوسطٍ *POP*. توحي هذه الأسماء بالمكذّسات الحقيقية كمكذّسات الصحن ذات النوايض المستخدمة في المقاهي. إن ترتيب سحب الصحن من المكثس يعاكس ترتيب إدراجها في المكثس، نظرًا لأننا نسحب فقط الصحن الموجود في أعلى المكثس لأنه الوحيد المتاح للسحب.

يبين الشكل 1.10 تنخيز مكثس يتألف من  $n$  عنصرًا على الأكثر باستخدام صيغة  $S[1..n]$ . للصيغة واصفةً  $S.top$  تعطي الدليل اللائق للعنصر المدرج آخرًا. يتألف المكثس من العناصر  $S[1..S.top]$  حيث  $S[1]$  هو العنصر الموجود في قعر المكثس، و  $S[S.top]$  العنصر الموجود في قمة المكثس.



**الشكل 1.10** تنجيز المكئس  $S$  باستخدام صفيقة. تُظهر عناصر المكئس فقط في اللواضع المظلمة تظليلاً خفيفاً. (أ) المكئس  $S$  وفيه 4 عناصر. العنصر الموجود في القمة هو 9. (ب) المكئس  $S$  بعد الاستدعاءات  $PUSH(S, 17)$  و  $PUSH(S, 3)$ . (ت) المكئس  $S$  بعد أن أعاد الاستدعاء  $POP(S)$  العنصر 3 وهو العنصر الذي دُفع به أخيراً في المكئس. ومع أن العنصر 3 لا يزال يظهر في الصفيقة، إلا أنه لم يعد في المكئس. فالعنصر الموجود في قمة المكئس هو العنصر 17.

إذا كان  $S.top = \square$ ، فإن هذا يعني أن المكئس لا يحتوي على عناصر وهو فارغ *empty*. يمكننا اختبار خلو المكئس باستخدام عملية الاستفسار  $STACK-EMPTY$ . إذا حاولنا نزع عنصرٍ من مكئس فارغ نقول عندها أن المكئس *يفيض* *underflows*، وهذا يعتبر بانطبع خطأً. وإذا تجاوز  $S.top$  القيمة  $n$  فإن المكئس *يفيض* *overflows*. (في تنجيزنا الوارد ضمن شبه الرماز لا نشتم بحالة فيض المكئس.) نستطيع تنجيز عمليات المكئس ببضعة أسطر من الرماز.

$STACK-EMPTY(S)$

```

1  if  $S.top == \square$ 
2      return TRUE
3  else return FALSE
    
```

$PUSH(S, x)$

```

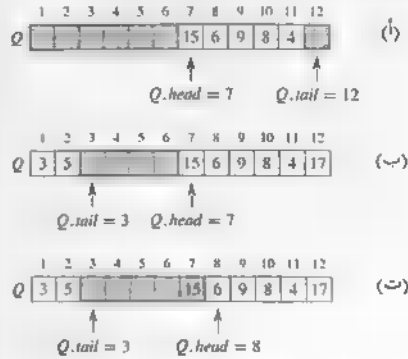
1   $S.top = S.top + 1$ 
2   $S[S.top] = x$ 
    
```

$POP(S)$

```

1  if  $STACK-EMPTY(S)$ 
2      error "underflow"
3  else  $S.top = S.top - 1$ 
4      return  $S[S.top + 1]$ 
    
```

بين الشكل 1.10 آثار عمليتي التعديل  $PUSH$  و  $POP$ . تستغرق كلُّ عمليةٍ من عمليات المكئس الثلاث زمناً  $O(1)$ .



**الشكل 2.10** تنجيز الرتل باستخدام صفيفة  $Q[1..12]$ . تظهر عناصر الرتل في المواضع المظلمة نظرياً صفيفاً فقط. (أ) في الرتل 5 عناصر، في المواضع  $Q[7..11]$ . (ب) تشكيلة الرتل بعد الاستدعاءات  $ENQUEUE(Q, 17)$  و  $ENQUEUE(Q, 3)$  و  $ENQUEUE(Q, 5)$ . (ت) تشكيلة الرتل بعد أن يعيد الاستدعاء  $DEQUEUE(Q)$  قيمة المفتاح 15 الذي كان سابقاً في رأس الرتل، وقد أصبحت القيمة الجديدة للرأس هي للمفتاح 6.

### الأوتال

نسقي العملية INSERT في الأوتال:  $ENQUEUE$ ، ونسقي العملية DELETE في الأوتال:  $DEQUEUE$ . وكما في العملية POP في المكثسات، لا تأخذ العملية  $DEQUEUE$  أي موسط. تعمل خاصية FIFO الرتل بعمل كشف من الزمانين الذين ينتظرون التسديد أمام الصندوق. وللرتل رأس  $head$  وذيل  $tail$ . عندما يلحق عنصر بالرتل يأخذ مكانه في ذيل الرتل، تماماً كما يحصل عندما يسطف الزبون الذي يصل في الآخر في نهاية الصف. أما العنصر الذي يُتْرَك من الرتل فهو دوماً العنصر الموجود في رأس الرتل، تماماً كالزبون الموجود في أول الصف الذي كان أكثر الزمانين انتظاراً.

يبين الشكل 2.10 إحدى طرائق تنجيز رتل من  $n-1$  عنصراً على الأكثر باستخدام صفيفة  $Q[1..n]$ . يزود الرتل بواصفة  $Q.head$  تعطي الدليل للموقع لرأس الرتل أو تشير إليه. وتدل الوصفة  $Q.tail$  على الموقع التالي الذي سيُدْرَج فيه العنصر الجديد الواصل إلى الرتل. توجد عناصر الرتل في المواضع:  $Q.tail - 1, \dots, Q.head + 1, Q.head$ ، حيث "تلتف" حول البداية بمعنى أن الموضع 1 يتبع مباشرة للموضع  $n$  في ترتيب دائري. فإذا أصبح  $Q.head = Q.tail$ ، فهذا يعني أن الرتل فارغ. في البداية يكون  $Q.head = Q.tail = 1$ . فإذا كان الرتل فارغاً وحاولنا نزع عنصر منه، فإن ذلك يتسبب في غيظ الرتل. وإذا كان  $Q.head = Q.tail + 1$  أو كان  $Q.head = 1$  و  $Q.tail = Q.length$  فهذا يعني أن الرتل ممتلئ، وأية محاولة لإلحاق عنصر به تتسبب في غيظ الرتل.



في الإجراءات: ENQUEUE و DEQUEUE اللذين نعرضهما هنا، أغفلنا عملية التحقق من وقوع الخطأ في حالتي الفيض أو الفيض. (يطلب إليك في التمرين 1.10-4 أن تضيف الرماز الذي يتحقق من الخطأ في هذين الشرطين.) يفترض شبه الرماز أن  $n = Q.length$ .

ENQUEUE( $Q, x$ )

```
1   $Q.tail = x$ 
2  if  $Q.tail == Q.length$ 
3       $Q.tail = 1$ 
4  else  $Q.tail = Q.tail + 1$ 
```

DEQUEUE( $Q$ )

```
1   $x = Q.head$ 
2  if  $Q.head == Q.length$ 
3       $Q.head = 1$ 
4  else  $Q.head = Q.head + 1$ 
5  return  $x$ 
```

يبين الشكل 2.10 نتائج العمليتين ENQUEUE و DEQUEUE. وتستغرق كلٌّ منهما زمناً  $O(1)$ .

## تمارين

### 1-1.10

باستخدام الشكل 1.10 نموذجاً، أوضح نتيجة كلِّ عملية وفق التالي:  $PUSH(S, 1)$ ,  $PUSH(S, 4)$ ,  $POP(S)$ ,  $PUSH(S, 8)$ ,  $POP(S)$ ,  $PUSH(S, 3)$ ،  $POP(S)$  على مكدس  $S$  فارغ ابتداءً وعُزِّن في الصفيفة  $S[1..6]$ .

### 2-1.10

اشرح كيفية تمييز مكدسين في صفيفة واحدة  $A[1..n]$  بحيث لا يفيض أيٌّ منهما إلا في الحالة التي يبلغ فيها مجموع العناصر في المكدسين معاً  $n$ . ينبغي أن تنفذ العمليتان PUSH و POP خلال زمن  $O(1)$ .

### 3-1.10

باستخدام الشكل 2.10 نموذجاً، أوضح نتيجة كلِّ عملية في المتاليات  $ENQUEUE(Q, 4)$  و  $ENQUEUE(Q, 1)$  و  $ENQUEUE(Q, 3)$  و  $DEQUEUE(Q)$  و  $ENQUEUE(Q, 8)$  و  $DEQUEUE(Q)$  على رتل  $Q$  فارغ ابتداءً وعُزِّن في الصفيفة  $Q[1..6]$ .

### 4-1.10

أعد كتابة ENQUEUE و DEQUEUE بحيث تكشف حصول الفيض والفيض في الرتل.

### 5-1.10

رأينا أن المكدس يسمح بإدراج العناصر وحذفها من طرف واحد فقط، وأن الرتل يسمح بالإدراج في أحد

الطرفين والحذف من الطرف الآخر، أما *deque* (وهو رتليّ ثنائي الطرفين) فإنه يُسمح بالإدراج والحذف من كلا الطرفين. اكتب أربعة إجراءات تستغرق زمنًا  $O(1)$  لإدراج العناصر وحذفها من طرفيّ رتليّ ثنائي الطرفين *deque* منخّزٍ باستخدام صفيفة.

### 6-1.10

بين كيف ننخّز رتلاً باستخدام مكّسّين. حلّّل زمن تنفيذ عمليات الرتل.

### 7-1.10

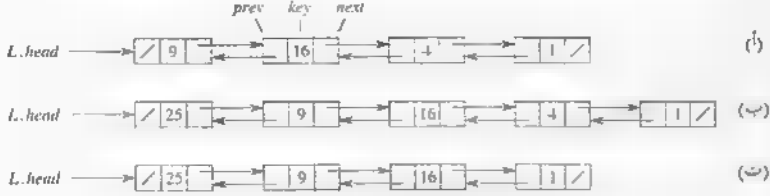
بيّن كيف ننخّز مكّسّاً باستخدام رتليّين. حلّّل زمن تنفيذ عمليات للمكّسّ.

## 2.10 اللوائح المترابطة

اللائحة المترابطة بنيتُ معطيات تُرتّب فيها الأغراض ترتيباً خطيّاً. ومع ذلك، تختلف اللائحة الخطية عن الصفيفة من جهة طريقة الترتيب؛ ففي الصفيفة يُحدّد دليل الصفيفة الترتيب الخطي، في حين يُحدّد الترتيب في اللائحة المترابطة باستخدام مؤشر في كل غرض. تقدّم اللوائح المترابطة تحليلاً بسيطاً ومرناً للمجموعات الديناميكية، وتدعم جميع العمليات المشار إليها في الصفحة 230 (وان لم تكن فعالة بالضرورة).

وكما يظهر في الشكل 3.10، فإن كلّ عنصر في *لائحة مضاعفة الترابط doubly linked list L* هو غرضٌ يمتلك الوصفة *key* وواصفين آخرين هما: *next* والسابق *prev*. ويمكن للفرض أن يحتوي أيضاً معطيات تابعة أخرى. ليكن  $\equiv$  عنصراً في اللائحة، يشير *x.next* إلى العنصر التالي في اللائحة المترابطة، ويشير *x.prev* إلى العنصر السابق له فيها. فإذا كان  $x.prev = \text{NIL}$  فهذا يعني أنه لا يوجد سابق للعنصر *x*، ومن ثمّ فهو العنصر الأول في اللائحة أو رأس *head* اللائحة. وإذا كان  $x.next = \text{NIL}$  فهذا يعني أنه لا يوجد عنصر يلي العنصر *x*، ومن ثمّ فهو العنصر الأخير في اللائحة أو ذيل *tail* اللائحة. تشير الوصفة *L.head* إلى العنصر الأول في اللائحة. فإذا كان  $L.head = \text{NIL}$ ، فإن اللائحة تكون فارغة.

نوجد عدة أشكال للوائح، فقد تكون اللائحة بسيطة الترابط أو مضاعفة الترابط، وقد تكون مرتبة أو غير مرتبة، وقد تكون دائرية أو غير دائرية. فإذا كانت اللائحة بسيطة الترابط *singly linked* فإننا نلغي المؤشر *prev* في كل عنصر. وإذا كانت اللائحة مرتبة *sorted* يكون الترتيب الخطي لللائحة مطابقاً للترتيب الخطي للمفاتيح المخزنة في عناصر اللائحة؛ وعندما يكون العنصر الأصغر رأساً لللائحة، والعنصر الأكبر ذيلاً لللائحة. وإذا كانت اللائحة غير مرتبة *unsorted* فيمكن أن تظهر العناصر في أي ترتيب. أما إذا كانت *اللائحة دائرية circular list* فإن للمؤشر *prev* في رأس اللائحة يشير إلى الذيل، والمؤشر *next* في ذيل اللائحة يشير إلى الرأس. وهكذا يمكن النظر إلى اللائحة وكأنها حلقة من العناصر. سنفترض في بقية هذا المقطع أن اللوائح التي نعامل معها هي لوائح غير مرتبة ومضاعفة الترابط.



**الشكل 3.10** (أ) لائحة مضاعفة الترابط  $L$  تمثل المجموعة الديناميكية  $\{1, 4, 9, 16\}$ . كل عنصر من اللائحة هو غرض يمتلك واصفة للمفتاح وواصفين للمؤشرين (يظهران على شكل أسهم) على الغرضين السابق والتالي. إن قيمة الواصفة  $next$  في الذيل والواصفة  $prev$  في الرأس هي  $NIL$  ويشار إليها بخط مائل. تشير الواصفة  $L.head$  إلى الرأس. (ب) بعد تنفيذ العملية  $LIST-INSERT(L, x)$  حيث  $x.key = 25$  يصبح في اللائحة المترابطة غرض جديد قيمة مفتاحه 25 ويأخذ مكانه كرأس جديد لللائحة. يشير الغرض الجديد إلى الرأس القديم ذي القيمة 9. (ت) النتيجة المترتبة عن الاستدعاء  $LIST-DELETE(L, x)$  حيث يشير  $x$  إلى الغرض ذي القيمة 4.

### البحث في اللائحة المترابطة

يعمل الإجراء  $LIST-SEARCH(L, k)$  على إيجاد أول عنصر يحوي لمفتاح  $k$  في اللائحة  $L$  وذلك بإجراء بحث خطي بسيط، ويعيد مؤشرًا إلى هذا العنصر. إن لم يجد الإجراء أي غرض في اللائحة يحوي المفتاح  $k$ ، فإنه يعيد  $NIL$ . بالعودة إلى اللائحة المترابطة في الشكل 3.10 (أ)، يعيد الاستدعاء  $LIST-SEARCH(L, 4)$  مؤشرًا إلى العنصر الثالث، ويعيد الاستدعاء  $LIST-SEARCH(L, 7)$  النتيجة  $NIL$ .

**LIST-SEARCH(L, k)**

```

1   $x = L.head$ 
2  while  $x \neq NIL$  and  $x.key \neq k$ 
3       $x = x.next$ 
4  return  $x$ 

```

يستغرق البحث في لائحة تتألف من  $n$  غرضًا باستخدام الإجراء  $LIST-SEARCH$  زمنًا  $\Theta(n)$  في أسوأ الحالات، نظرًا لأنه قد يحتاج إلى البحث في كامل اللائحة.

### الإدراج في اللائحة المترابطة

ليكن لدينا العنصر  $x$  الذي وُضعت في واصفته  $key$  قيمة معينة. يعمل الإجراء  $LIST-INSERT$  على "لصق" العنصر  $x$  في مقدمة اللائحة المترابطة كما يظهر في الشكل 3.10 (ب).

**LIST-INSERT(L, x)**

```

1   $x.next = L.head$ 
2  if  $L.head \neq NIL$ 

```

```

3   L.head.prev = x
4   L.head = x
5   x.prev = NIL

```

(تذكر أن التدوين المستخدم للواصفات يمكن أن يكون متسلسلاً، بحيث تعبر  $L.head.prev$  عن الوصفة  $prev$  في الغرض الذي يشر إليه  $L.head$ ). إن زمن تنفيذ LIST-INSERT في لائحة تتألف من  $n$  عنصراً هو  $O(1)$ .

### الحذف من اللائحة المترابطة

يُزيل الإجراء LIST-DELETE عنصراً  $x$  من اللائحة المترابطة  $L$ . ويجب أن يأخذ هذا الإجراء مؤشراً إلى  $x$  ثم "يفك لعنقه" من اللائحة ويحدث المؤشرات. إذا كنا نرغب في حذف عنصر ذي مفتاح محدد، فيجب أن نستدعي أولاً LIST-SEARCH لاسترداد المؤشر إلى هذا العنصر.

```

LIST-DELETE( $L, x$ )
1  if  $x.prev \neq NIL$ 
2      $x.prev.next = x.next$ 
3  else  $L.head = x.next$ 
4  if  $x.next \neq NIL$ 
5      $x.next.prev = x.prev$ 

```

يبين الشكل 3.10 (ت) كيف يُحذف عنصر من اللائحة المترابطة. يعمل LIST-DELETE في زمن  $O(1)$ . ولكن إذا كنا نرغب بحذف عنصر ذي مفتاح محدد، فيلزمنا  $\Theta(n)$  في أسوأ الحالات لأننا يجب أن نستدعي LIST-SEARCH أولاً لاكتشاف العنصر.

### الخرس

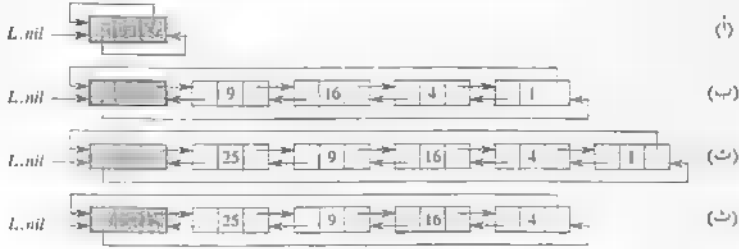
يمكن أن يصبح رماز الإجراء LIST-DELETE أبسط إذا استطعنا تجاهل الشروط الحدية على رأس وذيل اللائحة.

```

LIST-DELETE'( $L, x$ )
1   $x.prev.next = x.next$ 
2   $x.next.prev = x.prev$ 

```

الحارس *sentinel* غرضه شكلياً يمكننا من تبسيط الشروط على الحدود. لنفترض مثلاً أننا نزود اللائحة  $L$  بغرض  $L.nil$  يمثل  $NIL$ ، ولكنه يمتلك جميع الواصفات التي تمتلكها بقية عناصر اللائحة. وأينما ورد ذكر  $NIL$  في رماز اللائحة، نستعاض عنه بالحارس  $L.nil$ . يبين الشكل 4.10 كيف تتحول لائحة مضاعفة الترابط العادية إلى لائحة دائرية، مضاعفة الترابط مزودة بحارس *circular, doubly linked list with a sentinel*.



**الشكل 4.10** لائحة دائرية مضاعفة الترابط مزودة بحارس. يظهر الحارس  $L.nil$  بين الرأس والذيل. لم تعد هناك حاجة للموصلة  $L.head$ ، لأننا يمكن أن نشقُّ إلى رأس اللائحة عبر  $L.nil.next$ . (أ) لائحة فارغة. (ب) اللائحة المترابطة الواردة في الشكل 10.3 (أ)، مع المفتاح ■ في الرأس والمفتاح 1 في الذيل. (ت) اللائحة بعد تنفيذ  $LIST-INSERT(L, x)$ ، حيث  $x.key = 25$ . وقد أصبح الغرض الجديد رأساً لللائحة. (ث) اللائحة بعد حذف الغرض ذي المفتاح 1، وقد أصبح الغرض ذو المفتاح 24 ذيل اللائحة الجديد.

حيث يوضع الحارس  $L.nil$  بين الرأس والذيل. تشير الموصلة  $L.nil.next$  إلى رأس اللائحة، وتشير  $L.nil.prev$  إلى ذيلها. وبالمثل، فإن كلاً من الموصلتين  $next$  الخاصة بالذيل و  $prev$  الخاصة بالرأس تشير إلى  $L.nil$ . ولما كان  $L.nil.next$  يشير إلى الرأس، يمكننا حذف الموصلة  $L.head$  حذفاً كاملاً، والاستعاضة عنها أينما ورد ذكرها بـ  $L.nil.next$ . يبيِّن الشكل 4.10 (أ) أن اللائحة الفارغة تتألف من الحارس فقط، وأن كلاً من  $L.nil.next$  و  $L.nil.prev$  يشير إلى  $L.nil$ . يبقى رمز  $LIST-SEARCH$  كما كان سابقاً، مع تبديل  $NIL$  و  $L.head$  أينما ورد كما ذكرنا آنفاً.

$LIST-SEARCH'(L, k)$

```

1   $x = L.nil.next$ 
2  while  $x \neq L.nil$  and  $x.key \neq k$ 
3       $x = x.next$ 
4  return  $x$ 

```

نستخدم الإجراء  $LIST-DELETE'$  المؤلف من سطرين لحذف عنصر من اللائحة. ونستخدم الإجراء التالي لإدراج عنصر في اللائحة.

$LIST-INSERT'(L, x)$

```

1   $x.next = L.nil.next$ 
2   $L.nil.next.prev = x$ 
3   $L.nil.next = x$ 
4   $x.prev = L.nil$ 

```

يبين الشكل 4.10 أثر  $LIST-INSERT'$  و  $LIST-DELETE'$  على عيّنة من اللوائح.

نادراً ما يُخَفِّض الحرسُ الحدودَ للمقابلة لزمن العمليات على بي للعمليات، ولكنه قد يُخَفِّض المعاملات الثابتة. وينحصر الكسب الناتج عن استخدام الحرس في الحلقات عادةً في وضوح الرماز لا في السرعة. فمثلاً، يُسَيِّطُ استخدام الحرس الرماز الخاص باللائحة المترابطة، ولكننا نوفر هنا زمناً  $O(1)$  فقط في الإجراءات 'LIST-DELETE' و 'LIST-INSERT'. غير أن استخدام الحرس في حالات أخرى يساعد على إحكام الرماز في الحلقة، وهذا يُخَفِّض أمثال  $n$  أو  $n^2$  في زمن التنفيذ.

يجب أن تُستخدم الحرس بحذر، فإذا كانت لدينا عدة لوائح صغيرة، قد يشكّل الحزن الإضافي الذي يُستخدم لحرس هذه اللوائح ضياعاً مهماً في الذاكرة. في هذا الكتاب، نستخدم الحرس فقط عندما نجد أنه يَسَيِّطُ الرماز تبسيطاً حقيقياً.

## تعاريف

### 1-2.10

هل يمكن تمييز عملية INSERT الخاصة بالمجموعات الديناميكية في اللوائح المترابطة البسيطة بزمن  $O(1)$ ؟ وماذا عن DELETE؟

### 2-2.10

يُجَزَّزُ مكشّماً باستخدام لائحة مترابطة بسيطة  $L$ . يجب أن يبقى زمن العمليتين PUSH و POP  $O(1)$ .

### 3-2.10

يُجَزَّزُ رتلاً باستخدام لائحة مترابطة بسيطة  $L$ . يجب أن يبقى زمن العمليتين ENQUEUE و DEQUEUE  $O(1)$ .

### 4-2.10

مُرّ معنا أنفاً أن كلّ تكرارٍ للحلقة في الإجراء 'LIST-SEARCH' يحتاج إلى اختبارين: أحدهما للتأكد أن  $x \neq L.nil$ ، والآخر للتأكد أن  $x.key \neq k$ . يَبَيِّنُ كيف يمكننا حذف اختبار  $L.nil \neq x$  في كل تكرار.

### 5-2.10

يُجَزَّزُ العمليات المعجمية INSERT و DELETE و SEARCH مستخدماً لوائح مترابطة بسيطة ودائرية. ما هي أزمته تنفيذ هذه الإجراءات؟

### 6-2.10

تأخذ عملية UNION الخاصة بالمجموعات الديناميكية مجموعتين  $S_1$  و  $S_2$  منفصلتين دخالاً لها، وتعيد مجموعة  $S = S_1 \cup S_2$  مؤلفة من جميع عناصر  $S_1$  و  $S_2$ . تُعَدُّ هذه العملية المجموعتين  $S_1$  و  $S_2$  عادةً. يَبَيِّنُ كيف تتحمّل UNION زمناً  $O(1)$  باستخدام لائحة مناسبة باعتبارها بنية معطيات.

### 7-2.10

اكتب إجراء غير غوّدي يقلب لائحة مترابطة بسيطة مؤلفة من  $n$  عنصراً بزمن  $\Theta(n)$ . يجب ألا يُستخدم هذا الإجراء أماكن ثابتة للخرن أكثر مما تحتاجه اللائحة نفسها.

### 8-2.10 \*

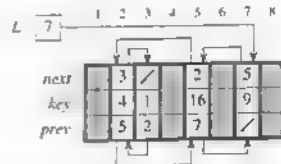
اشرح كيف تتخزّ اللوائح المترابطة المضاعفة مستخدماً فقط قيمةً واحدةً من نوع مؤشر  $x.np$  لكل عنصر بدلاً من استخدام المؤشرين للمعادين ( $prev$  و  $next$ ). افترض أن جميع قيم المؤشرات تكافئ أعداداً صحيحة ممثلة في  $k$  خانة ( $k$  - bit)، وعرف  $x.np$  بحيث يكون  $x.np = x.next \text{ XOR } x.prev$ ، أي "الخير المقصور" لـ  $k$  خانة ثنائية بين  $x.next$  و  $x.prev$ . (يعبر 0 عن القيمة NIL). تأكد أنك وضعت المعلومات اللازمة للنفاد إلى رأس اللائحة. بّن كيف تتخزّ العمليات SEARCH و INSERT و DELETE على هذه اللائحة، وكيف تُقلب هذه اللائحة بزمن  $O(1)$ .

## 3.10 تنجيز المؤشرات والأغراض

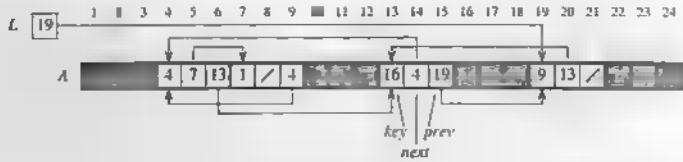
كيف نتخزّ المؤشرات والأغراض في اللغات التي ليس لديها مثل هذه الأنماط؟ سنرى في هذا المقطع طريقتين لتنجيز بني المعطيات المترابطة دون الاستخدام الصريح لنمط معطيات مؤشر. سنركّب الأغراض والمؤشرات انطلاقاً من الصفيفات وأدلة الصفيفات.

### تمثيل الأغراض بعدة صفيفات

نستطيع تمثيل مجموعة أغراض فتملك الواصفات ذاتها باستخدام صفيفة لكل واصفة. على سبيل المثال، يبين الشكل 5.10 تنجيز اللائحة المترابطة الواردة في الشكل 3.10 (أ) باستخدام ثلاث صفيفات. تحتفظ الصفيفة  $key$  بقيم المفاتيح الموجودة حالياً في المجموعة الديناميكية، وتخزّن المؤشرات في الصفيفتين  $prev$  و  $next$ . تمثّل عناصر الصفيفات  $key[x]$  و  $next[x]$  و  $prev[x]$  غرضاً واحداً في اللائحة المترابطة، وذلك لكل دليل صفيفة معطى  $x$ . ويوجب هذا التفسير، يكون المؤشر  $x$  ببساطة دليلاً مشتركاً للصفيفات  $key$  و  $next$  و  $prev$ .



**الشكل 5.10** اللائحة المترابطة الواردة في الشكل 3.10 (أ) ممثلة بالصفيفات  $key$  و  $next$  و  $prev$ . تمثّل كل شريحة عمودية من الصفيفات غرضاً واحداً. تتوافق المؤشرات المخزنة مع أدلة الصفيفات المبينة في الأعلى. وتوضح الأسهم كيف تفسر هذه الأدلة. تحتوي المواضع المظلة تظليلاً عميقاً عناصر اللائحة. ويحتفظ التحول  $L$  بالدليل الذي يدل على الرأس.



**الشكل 6.10** اللاتحة المترابطة الواردة في الشكلين 3.10(أ) و 5.10 ممثلة في صيغة وحيدة  $A$ . كل عنصر في اللاتحة هو غرض يشغل صيغة جزئية متلاصقة طولها 3 ضمن الصيغة. توافق الوصفات الثلاثة  $key$  و  $next$  و  $prev$  الريحانات  $\blacksquare$  و 1 و 2 على الترتيب ضمن كل غرض. إن مؤشرًا إلى الغرض هو دليل العنصر الأول في الغرض. طُلِّت الأغراض المخبئية على عناصر اللاتحة تظليلًا خفيًا، وتبين الأسهم الترتيب في اللاتحة.

في الشكل 3.10(أ) يتبع الغرض الذي يحمل القيمة 4 الغرض الذي يحمل القيمة 16 في اللاتحة المترابطة. أما في الشكل 5.10، فتظهر القيمة 4 في  $key[2]$  و القيمة 16 في  $key[5]$ ، وكذلك  $next[5] = 2$  و  $prev[2] = \blacksquare$ . ومع أن الثابت  $NIL$  يظهر في الوصفة  $next$  في ذيل اللاتحة وفي الوصفة  $prev$  في رأسها، فإننا نستعمل عادةً عددًا صحيحًا (مثل 0 أو -1) لا يمكن أن يعبر عن دليل فعلي في الصفقات. يُستخدم متحول  $L$  لحفظ الدليل الخاص برأس اللاتحة.

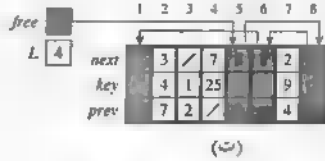
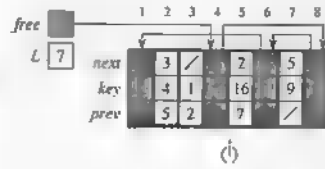
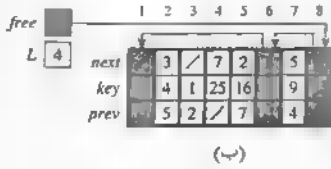
### تمثيل الأغراض بصيغة وحيدة

تُخزَّن الكلمات في ذاكرة الحواسيب عادةً باستخدام أعداد صحيحة تقع فيها بين  $\blacksquare$  و  $M-1$ ، حيث  $M$  عدد صحيح كبير مناسب. وفي العديد من لغات البرمجة يشغل الغرض مجموعة من المواقع المتلاصقة في ذاكرة الحاسوب. ويحتوي المؤشر على عنوان موضع الفكرة الأول من الغرض، أما المواضع الأخرى في الغرض نفسه فيمكن الحصول على دليلها بإضافة زيجان إلى المؤشر.

يمكننا أن نستخدم الاستراتيجية ذاتها لتحيز الأغراض في بيئات البرمجة التي لا توفر صراحةً أنماط معطيات مؤشر. فمثلًا يوضح الشكل 6.10 كيف يمكننا أن نستخدم صيغة وحيدة  $A$  لتخزين اللاتحة المترابطة التي رأيناها في الأشكال 3.10(أ) و 5.10. يشغل الغرض صيغة جزئية متلاصقة  $A[f..k]$ . وبوافق كل واصفة من الغرض زيجان تقع قيمته بين 0 و  $k-f$ ، ومؤشر على الغرض هو الدليل  $f$ . في الشكل 6.10 تكون قيم الريحان الموافقة لـ  $key$  و  $next$  و  $prev$  هي  $\blacksquare$  و 1 و 2 على الترتيب. ولقراءة قيمة  $l.prev$  اعتمادًا على مؤشر  $l$  معطى، نضيف القيمة  $l$  الخاصة بالمؤشر إلى قيمة الريحان 2، أي إننا نقرأ  $A[l+2]$ .

يعتبر التمثيل بصيغة واحدة مرئيًا من جهة أنه يسمح بتخزين الأغراض ذات الأطوال المختلفة في الصيغة نفسها. إن مسألة إدارة مجموعة الأغراض غير المتجانسة هذه أصعب من مسألة إدارة مجموعة





**الشكل 7.10** أثر الإجراءين **ALLOCATE-OBJECT** و **FREE-OBJECT**. (أ) اللاتحة الواردة في الشكل 5.10 (مطللة نظلياً عفيفاً) واللاتحة الحرة (مطللة نظلياً ثقيلاً). تبيّن الأسهم بنية اللاتحة الحرة. (ب) نتيجة استدعاء **ALLOCATE-OBJECT** (التي تعيد الدليل 4)، حيث تضع 25 في  $key[4]$  وتستدعي **LIST-INSERT(L, 4)**. إن الرأس الجديد للاتحة الحرة هو الغرض 8 الذي كان موافقاً لـ  $next[4]$  في اللاتحة الحرة. (ت) بعد تنفيذ **LIST-DELETE(L, 5)** نستدعي **FREE-OBJECT(5)**. يصبح الغرض 5 الرأس الجديد للاتحة الحرة، ويكون الغرض 7 التالي له في اللاتحة الحرة.

متجانسة من الأغراض تمتلك فيها جميع الأغراض الوصفات نفسها. ولما كانت معظم بنى المعطيات التي سندرسها تتألف من عناصر متجانسة، فيكفي أن نستخدم تمثيل الأغراض بعدة صيغيات لتحقيق هدفنا.

### تحصيل الأغراض وتحريرها

لكي نتمكن من إدراج مفتاح في مجموعة ديناميكية ممثلة بالاتحة مضاعفة الترابط، يجب أن نحصص مؤشراً لغرضي غير مستخدم حالياً في تمثيل اللاتحة للترابطة. لذا، يكون من المفيد إدراة تخزين الأغراض غير المستخدمة حالياً في تمثيل اللاتحة للترابطة بحيث نستطيع تخصيص أحدها عند الحاجة. نستخدم بعض الأنظمة **جميع نفايات garbage collector** ليكون مسؤولاً عن تحديد الأغراض غير المستخدمة، في حين تتحمل العديد من الأنظمة البسيطة مسؤولية إعادة الأغراض غير المستخدمة إلى المدير المسؤول عن الخزن. نستكشف هنا مسألة تخصيص وتحرير (أو فك تخصيص) الأغراض المتجانسة باستخدام مثال عن لاتحة مضاعفة الترابط ممثلة باستخدام عدة صيغيات.

لتفترض أن طول الصيغيات في التمثيل باستخدام عدة صيغيات هو  $m$ ، وأن المجموعة الديناميكية تحتوي، في لحظة ما،  $n \leq m$  عنصراً. إن هذا يعني أن هناك  $m - n$  عنصراً تمثل العناصر الموجودة حالياً في المجموعة



الشكل 8.10 لانتختان مترابطتان  $L_1$  (مظلة نظلياً خفيفاً) و  $L_2$  (مظلة نظلياً ثقيلاً) ولانحة حرة مرافقة لهما (الأشدّ نظلياً).

الديناميكية، أما الأغراض المتبقية وعددها  $m - n$  غرضاً فهي حرة *free*. يمكن أن تُستخدم الأغراض الحرة لتمثيل العناصر التي ستُدرج في المجموعة الديناميكية لاحقاً.

تُحفظ بالأغراض الحرة في قائمة أحادية الترابط نسبيها **اللائحة الحرة free list**. تُستخدم اللائحة الحرة العنيفة *next* فقط، التي تُعزّن المؤشرات *next* ضمن اللائحة. يحوي للتحويل العام *free* القيمة الدالة على رأس اللائحة الحرة. وعندما تكون المجموعة الديناميكية التي تمثلها اللائحة المترابطة  $L$  غير فارغة، يمكن أن تكون اللائحة الحرة ملازمة لللائحة  $L$  كما يظهر في الشكل 7.10. لاحظ أن كل غرض في التشيل إما أن يكون في اللائحة  $L$  وإما في اللائحة الحرة، ولا يمكن أن يكون فيهما معاً.

تتصرف اللائحة الحرة كمكدّس: فالغرض التالي المخصّص هو آخر غرض سبق تحريره. ويمكننا أن نُنجز عمليتي المكدّس *PUSH* و *POP* باستخدام اللائحة وذلك لتنحيز الإجراءات الخاصة بتخصيص الأغراض وتحريرها على الترتيب. نفترض أن للتحويل العام *free* المستخدم في الإجراءات التالية يشير إلى العنصر الأول من اللائحة الحرة.

```

ALLOCATE-OBJECT()
1  if free == NIL
2      error "out of space"
3  else x = free
4      free = x.next
5      return x
    
```

```

FREE-OBJECT(x)
1  x.next = free
2  free = x
    
```

في البداية، تحوي اللائحة الحرة جميع الأغراض غير المخصّصة وعددها  $n$ . وعندما تُستنفذ اللائحة الحرة، يُعلن الإجراء *ALLOCATE-OBJECT* حدوث خطأ. يمكننا استخدام لائحة حرة وحيدة لتخزين عدة لوائح مترابطة. يبيّن الشكل 8.10 لانتختين مترابطتين ولانحة حرة مرافقة لهما عبر الصفيفات *key* و *next* و *prev*.

يُنَفَّذُ الإجراءات في زمن  $O(1)$ ، وهو ما يجعلهما عمليتين تمامًا. ويمكن تعديلهما ليعملا مع أية مجموعة متجانسة من الأغراض، وذلك بجعل أي من واصفات الأغراض يتصرّف مثل الواصفة *next* في اللائحة الحرة.

### تمارين

#### 1-3.10

ارسم شكلاً لمتتالية العناصر: (13, 4, 8, 19, 5, 11) المخزّنة على أنها لائحة مضاعفة الترابط باستخدام تمثيل بعدة صيغيات، ثم كرّر الرسم باستخدام تمثيل بصيغة وحيدة.

#### 2-3.10

اكتب الإجرائين *ALLOCATE-OBJECT* و *FREE-OBJECT* في حالة مجموعة متجانسة من الأغراض، منخّرة باستخدام تمثيل بصيغة وحيدة.

#### 3-3.10

لماذا لا نحتاج إلى وضع قيمة (أو تخيئة) الواصفات *prev* في الأغراض عند تنحيز الإجرائين *FREE-OBJECT* و *ALLOCATE-OBJECT*؟

#### 4-3.10

في معظم الحالات، نرغب بأن تكون جميع عناصر اللائحة مضاعفة الترابط مترابطة *compact* في مكان الحزن، فنستخدم مثلاً المواقع ذات الأدلة *m* الأولى من التمثيل باستخدام عدة صيغيات. (وهذه هي حالة بيئة الحوسبة ذات الذاكرة الافتراضية الصفّحية *paged virtual-memory computing enviroment*). اشرح كيف يُنجز الإجراءات *ALLOCATE-OBJECT* و *FREE-OBJECT* بحيث يكون التمثيل مترابطاً. افترض أنه لا توجد مؤشرات إلى عناصر من اللائحة المترابطة خارج اللائحة نفسها (نفسه): استخدم تنحيز المكّنس باستخدام الصيغة).

#### 5-3.10

لنكن *L* لائحة مضاعفة طولها *n* مخزّنة في الصيغيات *key* و *prev* و *next* التي يبلغ طولها *m*. لنفترض أن الإجراءات *ALLOCATE-OBJECT* و *FREE-OBJECT* يديران هذه الصيغيات، وأنهما يحتفظان باللائحة حرة مضاعفة الترابط *F*. ولنفترض كذلك أنه من بين العناصر التي عددها *m*، يوجد *n* عنصراً فقط في اللائحة *L*، و *m - n* عنصراً في اللائحة الحرة. اكتب الإجراء *COMPACTIFY-LIST(L, F)* الذي يأخذ اللائحة *L* واللائحة الحرة *F*، وينقل عناصر *L* بحيث تشغل هذه العناصر مواقع الصيغة *1, 2, ..., n*، ويُضبط اللائحة الحرة *F* بحيث تبقى صحيحة، وبحيث تشغل المواقع *m, m + 1, n + 2, ..., n + 1* في الصيغة. يجب أن يكون زمن تنفيذ هذا الإجراء  $O(n)$ ، ويجب أن يُستخدم كمية ثابتة من المساحة الإضافية فقط. ناقش بشأن صحة إجرائك.

## 4.10 تمثيل الأشجار ذوات الجذور

يمكن تطبيق طرائق تمثيل اللوائح التي رأيناها في اللقطع السابق على أية بنية معطيات متجانسة. في هذا المقطع، ندرس على وجه التحديد مسألة تمثيل الأشجار ذوات الجذور باستخدام بني للمعطيات المترابطة. نبدأ أولاً بدراسة الأشجار الثنائية ثم نقدم طريقة للأشجار ذوات الجذور التي يكون للعقد فيها عدد اعتباري من الأبناء.

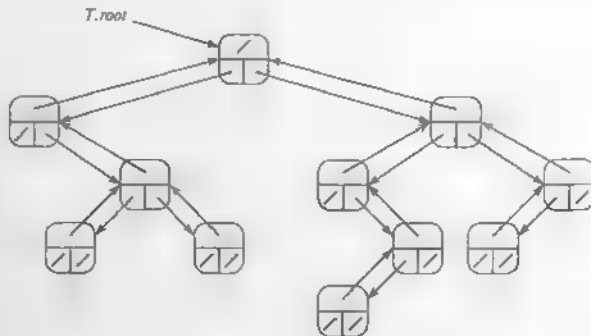
نمثل كل عقدة من الشجرة بغرض. وكما في اللوائح المترابطة، نفترض أن كل عقدة تحتوي الوصفة  $key$ . أما بقية الوصفات التي نحمنا، فهي مؤشرات إلى عقد أخرى، وتختلف تبعاً لنوع الشجرة.

## الأشجار الثنائية

يبيّن الشكل 9.10 كيف نستخدم الوصفات  $p$  و  $left$  و  $right$  لحزن مؤشرات إلى الأب، والابن الأيسر، والابن الأيمن، لكل عقدة في شجرة ثنائية  $T$ . إذا كان  $x.p = NIL$ ، فإن  $x$  يكون هو الجذر. وإذا لم يكن للعقدة  $x$  ابن أيسر، فإن  $x.left = NIL$ ، ومثل ذلك في حالة الابن الأيمن. يشار إلى جذر الشجرة  $T$  كلها بالوصفة  $T.root$ . فإذا كان  $T.root = NIL$  فهذا يعني أن الشجرة فارغة.

## الأشجار ذوات الجذور والتفرع غير المحدود

يمكن أن ينطبق منهج تمثيل الشجرة الثنائية على أي صف من الأشجار يكون فيه عدد أبناء كل عقدة ثابتاً ما  $k$  على الأكثر: نستعيض عن الوصفات  $left$  و  $right$  بـ  $child_1, child_2, \dots, child_k$ . لا يصلح هذا المنهج عندما يكون عدد أبناء العقدة غير محدود، مادامنا لا نعرف عدد الوصفات (لوائح) لعدد الصفات



الشكل 9.10 تمثيل الشجرة الثنائية  $T$ . تملك كل عقدة  $x$  الوصفات  $x.p$  (في الأعلى)، و  $x.left$  (في الأدنى يساراً)، و  $x.right$  (في الأدنى يميناً). لا تظهر هنا الوصفات  $key$ .

في التمثيل باستخدام عدة صفيفات) اللازم تخصيصه مقدّمًا. أضف إلى ذلك، أنه إذا كان عدد الأبناء  $k$  محدودًا بتابت كبير، وكان لدى معظم العقد عدد صغير من الأبناء، فإننا سنحسّر قدرًا كبيرًا من الذاكرة.

لحسن الحظ، يوجد منهج ذكي لتمثيل الأشجار ذوات العدد الاعتيادي من الأبناء. يتميز هذا المنهج باستخدام  $O(n)$  فقط من مساحة التخزين لأية شجرة ذات جذر فيها  $n$  عقدة. يبيّن الشكل 10.10 التمثيل باستخدام الابن الأيسر والشقيق الأيمن *left child, right sibling representation*. وكما رأينا سابقًا، تحتوي كل عقدة على مؤشر أبي  $p$ ، و  $T.root$  الذي يشير إلى جذر الشجرة  $T$ . وبدلاً من أن يكون لكل عقدة  $x$  مؤشر إلى كل ابن من أبنائها، يكون لها مؤشران فقط:

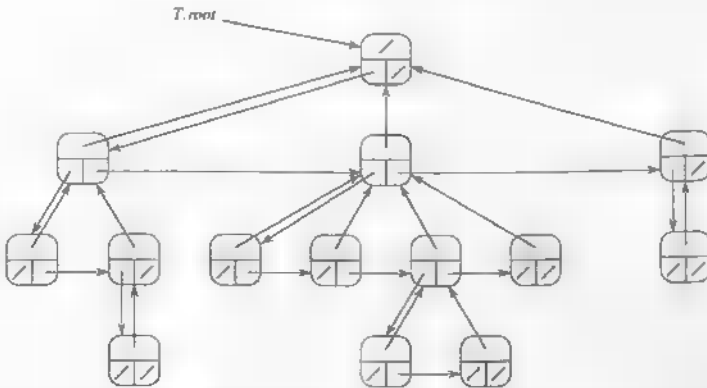
1. المؤشر  $x.left-child$  الذي يشير إلى ابن العقدة  $x$  الموجود في أقصى اليسار.

2. المؤشر  $x.right-sibling$  الذي يشير إلى شقيق  $x$  الموجود مباشرة إلى يمينه.

إذا لم يكن للعقدة  $x$  أبناء، يكون  $x.left-child = NIL$ ، وإذا كانت العقدة  $x$  هي الابن الموجود في أقصى اليمين، يكون  $x.right-sibling = NIL$ .

### طرق أخرى لتمثيل الأشجار

نلحاً أحياناً إلى تمثيل الأشجار ذوات الجذور بطرق أخرى. ففي الفصل 6 مثلاً، مثلنا الكومة - المعتمدة على شجرة ثنائية كاملة - باستخدام صفيفٍ وحيدة مع دليل العقدة الأخيرة في الكومة. وأما الأشجار التي تظهر في الفصل 21، فتعبّر بانجاء الجذر فقط، ولذلك لا يوجد سوى للمؤشرات الآباء؛ فلا وجود للمؤشرات إلى الأبناء. ولغة كثير من المناهج الممكنة الأخرى، تحدّد التطبيق أفضلها.



**الشكل 10.10** تمثيل الشجرة  $T$  باستخدام الابن الأيسر والشقيق الأيمن. تمتلك كل عقدة  $x$  الوصفيات  $x.p$  (في الأعلى)، و  $x.left-child$  (في الأدنى يساراً) و  $x.right-sibling$  (في الأدنى يمينا). لا تظهر هنا الوصفيات  $key$ .

## تمارين

## 1-4.10

ارسم الشجرة الثنائية التي يقع جذرها عند الدليل 6 والممتلئة بالواصفات التالية:

<i>right</i>	<i>left</i>	<i>key</i>	<i>index</i>
3	7	12	1
NIL	8	15	2
NIL	10	4	3
9	5	10	4
NIL	NIL	2	5
4	1	18	6
NIL	NIL	7	7
2	6	14	8
NIL	NIL	21	9
NIL	NIL	5	10

## 2-4.10

اكتب إجراء عؤدياً زمنه  $O(n)$  يطيع مفتاح كل عقدة في شجرة ثنائية معطاة ذات  $n$  عقدة.

## 3-4.10

اكتب إجراء غير عؤدي زمنه  $O(n)$  يطيع مفتاح كل عقدة في شجرة ثنائية معطاة ذات  $n$  عقدة. استخدم مكثداً باعتباره بنية معطيات مساعدة.

## 4-4.10

اكتب إجراء زمنه  $O(n)$  يطيع جميع المفاتيح في شجرة ذات جذر اعتباطية فيها  $n$  عقدة، علماً بأن الشجرة مخزنة وفق التمثيل باستخدام الابن الأيسر والشقيق الأيمن.

## \* 5-4.10

اكتب إجراء غير عؤدي زمنه  $O(n)$  يطيع مفتاح كل عقدة في شجرة ثنائية معطاة ذات  $n$  عقدة. لا تستخدم أكثر من مساحة تخزين إضافية ثابتة خارج الشجرة نفسها، ولا تغير الشجرة ولو مؤقتاً أثناء الإجراء.

## \* 6-4.10

يحتاج التمثيل باستخدام الابن الأيسر والشقيق الأيمن لشجرة ذات جذر اعتباطية إلى ثلاثة مؤشرات لكل عقدة: *left-child* و *right-sibling* و *parent*. يمكن الوصول من أية عقدة إلى أبيها وتحديدته في زمن ثابت، ويمكن الوصول إلى أبنائها وتحديدهم في زمن خطي متعلق بعدد الأبناء. يبين كيف نستخدم مؤشرين فقط وقيمة بوليانية واحدة في كل عقدة بحيث يمكن الوصول إلى أبي عقدة أو إلى جميع أبنائها وتحديدهم في زمن خطي متعلق بعدد الأبناء.

## مسائل

## 1-10 مقارنات بين اللوائح

ما هو زمن التنفيذ المقارب في أسوأ الحالات، لكل عملية من عمليات المجموعات الديناميكية المذكورة في كل من أنواع اللوائح الأربعة المذكورة في الجدول التالي:

مرتبة	غير مرتبة	مرتبة	غير مرتبة	
مضاعفة الترابط	مضاعفة الترابط	أحادية الترابط	أحادية الترابط	
				SEARCH( $L, k$ )
				INSERT( $L, x$ )
				DELETE( $L, x$ )
				SUCCESSOR( $L, x$ )
				PREDECESSOR( $L, x$ )
				MINIMUM( $L$ )
				MAXIMUM( $L$ )

## 2-10 الكومات القابلة للدمج باستخدام اللوائح المترابطة

تدعم الكومة القابلة للدمج *mergeable heap* العمليات التالية: MAKE-HEAP (التي تنشئ كومة قابلة للدمج فارغة) و INSERT و MINIMUM و EXTRACT-MIN و UNION.<sup>1</sup> بين كيف تُنفّذ الكومات القابلة للدمج باستخدام اللوائح المترابطة في كل من الحالات التالية. حاول أن تجعل كل عملية فعالة قدر المستطاع. حلّل زمن تنفيذ كل عملية بدلالة حجم المجموعة (المجموعات) الديناميكية التي تعمل عليها.

أ. اللوائح مرتبة.

ب. اللوائح غير مرتبة.

ت. اللوائح غير مرتبة، والمجموعات الديناميكية المطلوب دمجها منفصلة.

## 3-10 البحث في لائحة مترابطة مرتبة

اهتمّ التمرين 4-3.10 بكيفية الاحتفاظ بلائحة من  $n$  عنصراً بحيث تكون مترابطة في المواقع الـ  $n$  الأولى في

<sup>1</sup> لئلا نكون قد عرّفنا الكومة القابلة للدمج بحيث تدعم العمليتين MINIMUM و EXTRACT-MIN فبمكنا أن نسميها الكومة وفق الأصغر القابلة للدمج *mergeable min-heap*. وبالمقابل، لو كانت تدعم العمليتين MAXIMUM و EXTRACT-MAX لسميها الكومة وفق الأكبر القابلة للدمج *mergeable max-heap*.

صفيقة ما. نفترض هنا أن جميع المفاتيح متميزة، وأن اللاتحة المتراصة مرتبة أيضاً، أي إن  $key[i] < key[next[i]]$  لجميع قيم  $i = 1, 2, \dots, n$ ، حيث  $next[i] \neq NIL$ . وسنفترض أيضاً أن لدينا متحولاً  $L$  يحتوي دليل العنصر الأول في اللاتحة. ضمن هذه الفرضيات، يطلب أن تبين أن الخوارزمية ذات العشوائية المتضاعفة التالية يمكن أن تُستخدم للبحث ضمن اللاتحة في زمن متوقع  $O(\sqrt{n})$ .

COMPACT-LIST-SEARCH( $L, n, k$ )

```

1   $i = L$ 
2  while  $i \neq NIL$  and  $key[i] < k$ 
3       $j = \text{RANDOM}(1, n)$ 
4      if  $key[i] < key[j]$  and  $key[j] \leq k$ 
5           $i = j$ 
6          if  $key[i] == k$ 
7              return  $i$ 
8       $i = next[i]$ 
9  if  $i == NIL$  or  $key[i] > k$ 
10     return NIL
11 else return  $i$ 
```

لو تجاهلنا الأسطر 7-3 من الإجراء، نحصل على خوارزمية عادية للبحث في لاتحة مترابطة مرتبة، حيث يشير الدليل  $i$  إلى جميع المواقع في اللاتحة، كل بدوره. ينتهي البحث عندما "يسقط" الدليل  $i$  عن نهاية اللاتحة أو عندما يصبح  $key[i] \geq k$ . في هذه الحالة، إذا كان  $key[i] = k$  فمن الواضح أننا وجدنا مفتاحاً قيمته  $k$ . أما إذا أصبح  $key[i] > k$ ، فعندها، لن نجد أبداً مفتاحاً قيمته  $k$ ، ومن ثم يكون التوقف عن البحث هو العمل المناسب.

تحاول الأسطر 7-3 أن تقفز قُدماً نحو موضع  $j$  اختير عشوائياً. تفيدينا هذه القفزة إذا كان  $key[j]$  أكبر من  $key[i]$  ولكنه ليس أكبر من  $k$ ؛ في مثل هذه الحالة، يعين  $j$  موقفاً في اللاتحة كان  $i$  سيبلغه أثناء عملية البحث العادية في اللاتحة. ونظراً لأن اللاتحة مترابطة، فإننا نعلم أن أي اختيار لـ  $j$  بين 1 و  $n$  سيطينا دليلاً لفرضي ما في اللاتحة وليس لفراغ في اللاتحة الحرة.

وبدلاً من أن نحمل أداء COMPACT-LIST-SEARCH مباشرة، فإننا سنحلل خوارزمية مرتبطة بها وهي: 'COMPACT-LIST-SEARCH' التي تتخذ حلقتين مستقلتين. تأخذ هذه الخوارزمية موسطاً إضافياً  $t$  يحدد حداً أعلى لعدد تكرارات الحلقة الأولى.

COMPACT-LIST-SEARCH'( $L, n, k, t$ )

```

1   $i = L$ 
2  for  $q = 1$  to  $t$ 
3       $j = \text{RANDOM}(1, n)$ 
4      if  $key[i] < key[j]$  and  $key[j] \leq k$ 
```



```

5      i = j
6      if key[i] == k
7          return i
8  while i ≠ NIL and key[i] < k
9      i = next[i]
10 if i == NIL or key[i] > k
11     return NIL
12 else return i

```

لمقارنة تنفيذ الخوارزميتين COMPACT-LIST-SEARCH( $L, n, k, t$ ) و COMPACT-LIST-SEARCH'( $L, n, k, t$ ) نفترض أن متتالية الأعداد الصحيحة التي تنتج عن استدعاءات RANDOM( $1, n$ ) هي نفسها في الخوارزميتين.

أ. افترض أن COMPACT-LIST-SEARCH( $L, n, k$ ) يحتاج  $t$  تكرارًا لحلقة while الواردة في الأسطر 2-8. علّل كون COMPACT-LIST-SEARCH'( $L, n, k, t$ ) تعيد نفس النتيجة، وأن العدد الكلي للتكرارات في كلٍّ من حلقتي for و while ضمن COMPACT-LIST-SEARCH' هو  $t$  على الأقل.

في استدعاء COMPACT-LIST-SEARCH'( $L, n, k, t$ )، ليكن  $X_t$  انتحول العشوائي الذي يصف المسافة في اللاتحة المترابطة (عبر سلسلة المؤشرات  $next$ ) بين الموقع  $i$  والموقع الموافق للمفتاح  $k$  المراد الوصول إليه بعد  $t$  تكرارًا في حلقة for في الأسطر 2-7.

ب. برّر أن الزمن المتوقع لتنفيذ COMPACT-LIST-SEARCH'( $L, n, k, t$ ) هو  $O(t + E[X_t])$ .

ت. بيّن أن  $E[X_t] \leq \sum_{r=1}^n (1 - r/n)^t$ . (لمصيح: استخدم المعادلة 25.2).

ث. بيّن أن  $\sum_{r=0}^{n-1} r^t \leq n^{t+1}/(t+1)$ .

ج. برهن أن  $E[X_t] \leq n/(t+1)$ .

ح. بيّن أن COMPACT-LIST-SEARCH'( $L, n, k, t$ ) تنقذ في زمن متوقع  $O(t + n/t)$ .

خ. استنتج أن COMPACT-LIST-SEARCH تنقذ في زمن متوقع  $O(\sqrt{n})$ .

د. لماذا نفترض أن جميع المفاتيح في COMPACT-LIST-SEARCH متمايزة فيما بينها؟ برّر لماذا لا تساعد القفزات العشوائية في المقارنة بالضرورة عندما تحتوي اللاتحة على قيم مكررة للمفاتيح.

## ملاحظات الفصل

نُعدُّ كُتب Aho و Hopcroft و Ullman [6] و knuth [209] مراجع ممتازة لبنى المعطيات البدائية. تشمل كتبٌ أخرى بنى للمعطيات الأساسية وتحيزها في لغة برمجة محددة. نذكر أمثلةً على هذا النوع من الكتب

الجامعية Goodrich و Tamassia [147] و Main [241] و Shaffer [311] و Weiss [352, 353, 354].  
 يقدّم Gonnet [145] معطياتٍ تجريبيةً تتعلق بأداء العديد من عمليات بنى للمعطيات.  
 إن أصول المكّدسات والأرتال باعتبارها بنى معطيات في علم الحاسوب غير واضحة، لأن المفاهيم الموافقة لها في الرياضيات والممارسات الورقية للأعمال كانت موجودة قبل إدخال الحواسيب الرقمية. ينوّه Knuth [209] العالم A. M. Turing بتطويره المكّدسات لربط المسافات الفرعية عام 1947.  
 ويبدو أن بنى للمعطيات المعتمدة على المؤشرات هي من اختراع الناس. فوفقاً لـ Knuth، يبدو أنه قد جرى استخدام المؤشرات في الحواسيب الأولى مع الذواكر الأسطوانية، وقد مثّلت لغة A-1 التي طوّرها G. M. Hopper في 1951 الصيغ الجبرية باعتبارها أشجاراً ثنائية. ويعترف Knuth بفضل لغة IPL-II التي طوّرها A. Newell و J. C. Shaw و H. A. Simon عام 1956 لاعترافها بأهمية استخدام المؤشرات والتشجيع على استخدامها. وقد تضمنت لغة IPL-III للطرّة في 1957 عمليات للكّنس صراحة.

تتطلب الكثير من التطبيقات مجموعة ديناميكية من المعطيات تدعم العمليات المعقدة فقط: INSERT و SEARCH و DELETE. فمثلاً، يحتفظ مترجم لغة برمجة ما بجدول رموز، مفاتيح العناصر فيه هي متتاليات محرفية اعتباطية تقابل المخرجات في اللغة. إن جدول التلييد هو بنية معطيات فعالة لتتبع المعاجم. ومع أن البحث عن عنصر ما في جدول تلييد يمكن أن يستغرق عملياً زمناً مماثلاً لزمّن البحث عن عنصر في لائحة مترابطة  $O(n)$  في أسوأ الحالات، فإن أداء التلييد يُقدّر جيداً جداً. بفرضيات معقولة، يصبح الزمن الوسطي للبحث عن عنصر ما في جدول تلييد هو  $O(1)$ .

يُعَمِّم جدول التلييد المفهوم الأبسط للصفيفة العادية. تُستخدَم العنونة المباشرة في الصفيفة العادية بفعالية قدرتنا على فحص أيّ موقع في صفيفة ما، في زمن  $O(1)$ . يناقش المقطع 1.11 العنونة المباشرة بتفصيل أكبر. وبمكثنا الاستفادة من العنونة المباشرة عندما يكون متاحاً لنا تخصيص صفيفة تحتوي على موضع واحد لكل مفتاح ممكن.

حين يكون عدد المفاتيح المخزّنة فعلياً صغيراً بالنسبة إلى العدد الكلي للمفاتيح الممكنة، تصبح جداول التلييد بديلاً فعالاً للعنونة المباشرة لصفيفة ما، لأن جدول التلييد يُستخدم عادةً صفيفة متناسب حجمها طرّاً مع عدد المفاتيح المخزّنة فعلياً. عوضاً عن استخدام المفتاح مباشرة باعتباره دليل صفيفة، يجري حساب دليل الصفيفة من المفتاح. يقدّم المقطع 2.11 الأفكار الرئيسة مع التركيز على "السلسلة chaining" باعتبارها طريقة لمعالجة "التصادمات collisions"، التي يقابل فيها أكثر من مفتاح دليل الصفيفة نفسه. أما المقطع 3.11 فيصف كيف يمكننا حساب دلائل الصفيفة من المفاتيح، باستخدام دوال التلييد. ستقدّم عدة متغيرات للموضوع الأساسي ونحلّها. يلقي المقطع 4.11 نظرة على "العنونة المفتوحة open addressing" التي هي طريقة أخرى لمعالجة التصادمات. وخلاصة القول هي أن التلييد تقانة فعالة جداً وعملية جداً: تتطلب العمليات المعقدة الأساسية وسطياً زمناً  $O(1)$  فقط. يشرح المقطع 5.11 كيف يستطيع "التلييد التام perfect hashing" دعم عمليات البحث في أسوأ الحالات بزمن  $O(1)$ ، حين تكون مجموعة المفاتيح المخزّنة ساكنة (أي حين لا تتغير مجموعة المفاتيح المخزّنة إطلاقاً بعد تخزينها).

## 1.11 جداول العنوان المباشر

العنونة المباشرة تقانة بسيطة تعمل جيداً حين تكون مجموعة المفاتيح الكلية  $U$  صغيرة على نحو معقول. افترض أن تطبيقاً ما، يحتاج إلى مجموعة ديناميكية لكل عنصر منها مفتاح مسحوب من المجموعة الكلية  $U = \{0, 1, \dots, m-1\}$ ، حيث  $m$  ليست كبيرة جداً، وسنفترض أنه لا يمتلك عنصران المفتاح نفسه.

لتمثيل المجموعة الديناميكية، نستخدم صفيقة، أو جدول عنوان مباشر *direct-address table*، يرمز إليه بـ  $T[0..m-1]$ ، ويوافق كل موقع أو شُكْب *slot* في هذا الجدول مفتاحاً من المجموعة الكلية  $U$ . يوضح الشكل 1.11 هذه للمنهية؛ حيث يُوْشر الشُكْب  $k$  إلى عنصر في المجموعة مفتاحه  $k$ . إذا كانت المجموعة لا تحتوي على عنصر مفتاحه  $k$ ، عندها يكون  $T[k] = \text{NIL}$ . تُنْخِز العمليات المعتمدة ببساطة.

DIRECT-ADDRESS-SEARCH( $T, k$ )

1 return  $T[k]$

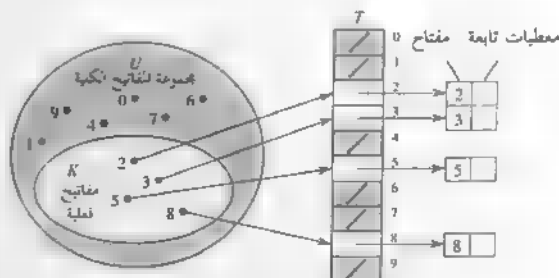
DIRECT-ADDRESS-INSERT( $T, x$ )

1  $T[x.\text{key}] = x$

DIRECT-ADDRESS-DELETE( $T, x$ )

1  $T[x.\text{key}] = \text{NIL}$

وتتطلب كلٌّ من هذه العمليات فقط زمناً  $O(1)$ .



**الشكل 1.11** تخزين مجموعة ديناميكية باستخدام جدول عنوان مباشر  $T$ . يوافق كل مفتاح في المجموعة الكلية  $U = \{0, 1, \dots, 9\}$  دليلاً في الجدول. تُحدّد مجموعة المفاتيح الفعلية  $k = \{2, 3, 5, 8\}$  الشُكُوب التي تحتوي مؤشرات على عناصر في الجدول. الشُكُوب الأخرى، للظلال بالفاصل تحتوي  $\text{NIL}$ .

في بعض التطبيقات، يمكن أن يحتوي جدول العنوان المباشر نفسه عناصر المجموعة الديناميكية. ذلك أنه عوضاً عن تخزين مفتاح عنصر ما والمعطيات التابعة له، في غرض خارج جدول العنوان المباشر مع مؤشر من الشُّب في الجدول إلى الغرض، نستطيع تخزين الغرض نفسه في الشُّب، وبذلك ندخر فضاء ذاكرة. وقد نستخدم مفتاحاً خاصاً ضمن الغرض للإشارة إلى شُّب فارغ. يضاف إلى ذلك أنه ليس من الضروري، في غالب الأحيان، تخزين مفتاح الغرض، لأنه إذا توفر لدينا دليل الغرض في الجدول، أمكننا الحصول على مفتاحه. ولكن، إذا لم تكن المفاتيح مخزنة، وجب أن تتوفر طريقة نتمكن من معرفة كون الشُّب فارغاً.

### تمارين

#### 1-1.11

افترض أن مجموعة ديناميكية  $S$  ممثلة بجدول عنوان مباشر  $T$  طوله  $m$ . صيغ الإرجاع الذي يكشف العنصر الأعظمي في  $S$ . ما هو أداء إرجاعك في أسوأ الحالات؟

#### 2-1.11

شعاع البتات *bit vector* هو ببساطة صفيقة من البتات (أصفار ووحدان). يشغل شعاع بتات طوله  $m$  فراغاً في الذاكرة أقل بكثير من صفيقة مكونة من  $m$  مؤشرًا. اشرح كيفية استخدام شعاع بتات لتمثيل مجموعة ديناميكية من العناصر المتميزة دون معطيات تابعة لها. يجب أن تنفذ العمليات المعجمية في زمن  $O(1)$ .

#### 3-1.11

اقترح طريقة لتتبع جدول عنوان مباشر لا تحتاج فيه مفاتيح العناصر المخزنة إلى أن تكون متميزة، ويمكن للعناصر أن تحتوي معطيات تابعة لها. يجب أن تنفذ العمليات المعجمية الثلاث (DELETE و INSERT و SEARCH) في زمن  $O(1)$ . (لا تنس أن عملية DELETE تعتبر للمؤشر - وليس للمفتاح - هو الذي يُحدّد الغرض المطلوب حذفه.)

#### ★ 4-1.11

نرغب بتحقيق معجم باستخدام عنوان مباشرة على صفيقة ضخمة. في البداية، قد تحتوي عناصر الصفيقة فضلات معطيات، واستبداء الصفيقة غير عملي بسبب حجمها. صيغ طريقة لتتبع معجم عنوان مباشر على صفيقة ضخمة. يجب أن يشغل كل غرض مخزن حجم ذاكرة  $O(1)$ ؛ يجب أن تأخذ كل من عمليات INSERT و DELETE و SEARCH زمنًا  $O(1)$ ؛ ويجب أن يستغرق استبداء بنية المعطيات زمنًا  $O(1)$ . (لمصيح: استخدم صفيقة إضافية، تُعالج باعتبارها مكثفًا حجمه هو عدد المفاتيح المخزنة فعليًا في المعجم، وذلك للمساعدة في تحديد كون عنصر ما في الصفيقة الضخمة ما يزال مستخدمًا أم لا.)

## 2.11 جداول التلييد

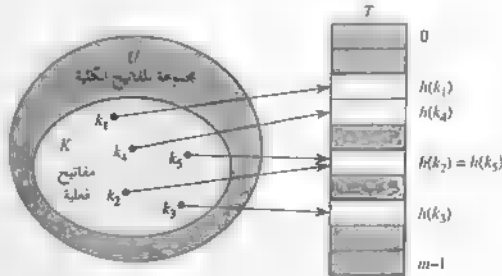
من المتالب الواضحة للعنونة المباشرة: إذا كانت المجموعة الكلية  $U$  كبيرة، فقد يكون من غير العملي، أو حتى للمستحيل، تخزين جدول  $T$  حجمه  $|U|$ ، إذا أخذنا بالحسبان حدود الذاكرة المتاحة في حاسوب عادي. إضافة إلى أن مجموعة المفاتيح  $K$  المخزنة فعلياً قد تكون صغيرة جداً مقارنة بـ  $U$ ، بحيث يضيع معظم الفراغ المخصص للجدول  $T$ .

إذا كانت مجموعة للمفاتيح  $K$  المخزنة في المعجم أصغر بكثير من المجموعة الكلية  $U$  لجميع المفاتيح الممكنة، تتطلب جدول التلييد ذاكرة أقل بكثير مما يتطلبه جدول العنوان المباشر. وبعبارة أدق، يمكننا أن نخفض متطلبات الذاكرة إلى حجم  $\Theta(|K|)$  في حين نحافظ على قاعدة أن يبقى زمن البحث عن عنصر ما في جدول تلييد هو  $O(1)$  فقط. إلا أن هذا الحد يتعلق بالزمن في الحالة الوسطى، على حين ينطبق على زمن البحث في أسوأ الحالات في حال العنونة المباشرة.

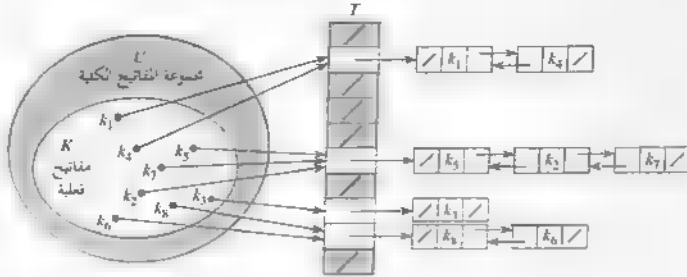
في العنونة المباشرة، يُخزّن عنصر مفتاحه  $k$  في الشق  $k$ . أما في التلييد، فيخزّن هذا العنصر في الشق  $h(k)$ ؛ حيث، نستخدم دالة تلييد  $h$  hash function لحساب شق المفتاح  $k$ . هنا يقابل  $h$  بين مجموعة المفاتيح الكلية  $U$  وشقوب جدول التلييد  $hash\ table$   $T[0..m-1]$ :

$$h : U \rightarrow \{0, 1, \dots, m-1\},$$

حيث يكون حجم جدول التلييد  $m$  عادةً أصغر بكثير من  $|U|$ . ونقول إن العنصر ذا المفتاح  $k$  يتليد  $hashes$  في الشق  $h(k)$ ؛ ونقول كذلك إن  $h(k)$  هي قيمة تلييد  $hash\ value$  المفتاح  $k$ . يوضح الشكل 2.11 الفكرة الرئيسة. إن دالة التلييد تخفض مجال دلائل الصفيقة، ومن ثم حجم الصفيقة. فوضاً عن الحجم  $|U|$ ، قد تأخذ الصفيقة الحجم  $m$ .



الشكل 2.11 استخدام دالة تلييد  $h$  لمقابلة مفاتيح بشقوب جدول تلييد. ولما كان المفتاحان  $k_2$  و  $k_5$  يقابلان الشق نفسه، فإنهما يتصادمان.



**الشكل 3.11** حل التصادم بالسلسلة. يحتوي كل شُـب  $T[j]$  من جدول التلييد لائحة مترابطة مؤلفة من جميع المفاتيح التي قيمة تلييدها  $j$ . مثلاً،  $h(k_1) = h(k_4)$  و  $h(k_2) = h(k_7) = h(k_8)$ . يمكن أن تكون اللائحة المترابطة بسيطة الترابط أو مضاعفة الترابط، ونظهرها هنا على أنها مضاعفة الترابط لأن الهدف يكون أسرع في هذه الحالة.

لغة عقبة واحدة: وهي أنه قد يتألف مفتاحان في الشُـب نفسه. نسمي هذه الحالة **تصادماً collision**. ولحسن الحظ، لدينا تقانات فعالة لإزالة التضارب الناتج عن التصادمات.

قد يكون الحل المثالي طبعاً هو في تجنب إحداث تصادمات. وقد نحاول تحقيق هذا الهدف باختيار دالة تلييد مناسبة  $h$ . إحدى الأفكار المطروحة هي جعل  $h$  تبدو "عشوائية"، وبذلك نتجنب التصادمات أو على الأقل نجعل عددها أصغر. يستحضر مصطلح "تلييد" صوراً من الخلط والتقطيع العشوائيين، نجعله يستحوذ على روح هذا النهج. (طبعاً، يجب أن تكون دالة التلييد  $h$  حتمية بحيث ينتج دخل معطى  $k$  المخرج  $h(k)$  نفسه دائماً). ولما كان  $|U| > m$ ، وجب أن يملك مفتاحان على الأقل قيمة التلييد نفسها؛ لذلك من المستحيل تجنب جميع التصادمات. وهكذا، ومع أن دالة التلييد المعسّمة جيداً تبدو "عشوائية" تخفّف عدد التصادمات إلى الحد الأدنى، فإن الحاجة إلى طريقة لتمييز التصادمات التي تحدث تبقى قائمة.

يقدم القسم المتبقي من هذا المقطع أبسط تقانة لتمييز التصادم، تسمى السلسلة. ويقدم المقطع 4.11 طريقة بديلة لتمييز التصادمات، تدعى العنونة المفتوحة.

### تمييز التصادم باستخدام السلسلة

في **السلسلة chaining**، نضع جميع العناصر التي تتّـبـد في نفس الشُـب في اللائحة المترابطة نفسها، كما يبين ذلك الشكل 3.11. يحتوي الشُـب  $j$  مؤشراً على رأس لائحة جميع العناصر المخزنة التي تلييدها  $j$ ؛ وإذا لم توجد مثل هذه العناصر، يحتوي الشُـب  $j$  القيمة NIL.

يمكن تبسيط العمليات المعجمية على جدول التلييد  $T$  بسهولة حين يجري تمييز التصادمات باستخدام السلسلة.

CHAINED-HASH-INSERT( $T, x$ )

1 insert  $x$  at the head of list  $T[h(x.key)]$

CHAINED-HASH-SEARCH( $T, x$ )

1 search for an element with key  $k$  in the list  $T[h(k)]$

CHAINED-HASH-DELETE( $T, x$ )

1 delete  $x$  from the list  $T[h(x.key)]$

إن زمن تنفيذ عملية الإدراج في أسوأ الحالات هو  $O(1)$ . وإن إجراء الإدراج سريع إلى حد ما، لأنه يفترض أن العنصر  $x$  الذي هو في قيد الإدراج غير موجود أصلاً في الجدول، ويمكننا أن نعتبر هذا الافتراض عند الضرورة (بكلفة إضافية) بالبحث قبل الإدراج عن العنصر ذي المفتاح  $x.key$ . وأما زمن التنفيذ في أسوأ الحالات لعملية البحث، فيتناسب مع طول اللاتحة؛ وسوف نحلل هذه العملية لاحقاً بعق أكثر. يمكننا أن نحذف عنصراً  $x$  في زمن  $O(1)$  إذا كانت اللوائح مضاعفة الترابط، كما هو موضح في الشكل 3.11. (لاحظ أن CHAINED-HASH-DELETE يأخذ في الدخول عنصراً  $x$  وليس مفتاحه  $k$ ، لذا فإننا لا نحتاج إلى البحث عن  $x$  أولاً. فإذا كان جدول التلييد يدعم الحذف، عندها يجب أن تكون لوائحه للترابط مضاعفة الترابط بحيث نستطيع حذف حدًا ما بسرعة. أما إذا كانت اللوائح بسيطة الترابط فقط - عند حذف عنصر ما  $x$  - ونحتاج علينا أولاً أن نجد في اللاتحة  $T[h(x.key)]$  كي نستخدم تحديث الواسطة  $next$  للعنصر السابق للعنصر  $x$ . هذا وتساوى أزمة التنفيذ المقارنة لكل من الحذف والبحث في اللوائح البسيطة الترابط.)

### تحليل التلييد مع السلسلة

ما هي جودة أداء التلييد مع السلسلة؟ وعلى وجه الخصوص، كم يستغرق البحث عن عنصر بدلالة مفتاح معطى؟

ليكن لدينا جدول تلييد  $T$  مكون من  $m$  سقياً ونحزّن  $n$  عنصراً، نعرف عامل التحميل  $\alpha$  load factor لـ  $T$  بأنه  $n/m$ ، وهو العدد الوسطي للعناصر المخزنة في سلسلة. سيكون تحليلنا بدلالة  $\alpha$ ، التي يمكن أن تكون أصغر من 1، أو مساوية له، أو أكبر منه.

إن سلوك التلييد مع السلسلة في أسوأ الحالات رديء جداً، حيث تطلب المفاتيح  $n$  كلها في الشقبة نفسه، مشكلةً لاتحة طوله  $n$ . وبذلك يكون زمن البحث في أسوأ الحالات  $\Theta(n)$  إضافةً إلى زمن حساب دالة التلييد - وهو ليس أفضل مما لو استخدمنا لاتحة واحدة لكل العناصر. من الواضح أننا لا نستخدم



جداول التلييد في حال أدائها في أسوأ الحالات. (يوفر التلييد الكامل، المشرح في المقطع 5.11، أداءً جيدًا في أسوأ الحالات حين تكون مجموعة المفاتيح ساكنة.)

يعتمد أداء الحالة الوسطى للتلييد على مدى الجودة التي تُوزَّع بها دالة التلييد  $h$  المفاتيح الواجب تخزينها بين  $m$  شُكْبًا، وسيطًا. يناقش للمقطع 3.11 هذه النقاط، لكننا سنفترض الآن أن أيَّ عنصرٍ معطى متساوي الأُرجحية في أن يُلَبَد في أيٍّ من الشُكُوب  $m$ ، بقطع النظر عن مكان تلييد أي عنصرٍ آخر. نسمي هذا الافتراض **التلييد المنتظم البسيط** *simple uniform hashing*.

نرمز إلى طول اللائحة  $[T[j]]$  بـ  $n_j$ ، لكل  $j = 0, 1, \dots, m-1$ . ومن ثمَّ يكون

$$n = n_0 + n_1 + \dots + n_{m-1}, \quad (1.11)$$

والتوقع لـ  $n_j$  هو  $E[n_j] = \alpha = n/m$ .

نفترض أن الزمن  $O(1)$  يكفي لحساب قيمة التلييد  $h(k)$ ، بحيث يتعلَّق الزمن الذي يتطلبه البحث عن عنصرٍ مفاتيحه  $k$  عطيًّا بـ  $n_{h(k)}$  طول اللائحة  $[T[h(k)]]$ . بإهمال الزمن  $O(1)$  اللازم لحساب دالة التلييد والنفاذ للشُكْب  $h(k)$ ، سندرس العدد المتوقع للعناصر التي تفحصها خوارزمية البحث، وهو عدد عناصر اللائحة  $[T[h(k)]]$  التي تفحصها الخوارزمية لمعرفة أيٍّ من مفاتيحها يساوي  $k$ . سنأخذ بالحسبان حالتين: في الحالة الأولى لا ينجح البحث: أي لا يوجد في الجدول أيُّ عنصرٍ قيمةً مفتاحه  $k$ . وفي الحالة الثانية ينجح البحث في إيجاد عنصرٍ مفاتيحه  $k$ .

### مبرهنة 1.11

يستغرق البحث غير الناجح في جدول تلييد جرى فيه حل التصادمات باستخدام الشُكُوب، زمنًا في الحالة الوسطى  $\Theta(1 + \alpha)$ ، وذلك بفرض أن التلييد منتظم بسيط.

**البرهان** بافتراض أن التلييد منتظم بسيط، فإن أيَّ مفتاح  $k$  غير مخزَّن في الجدول يمكن أن يُلَبَد في أيٍّ من الشُكُوب  $m$  باحتمال متساوٍ. وإن الزمن المتوقع للبحث غير الناجح لأي مفتاح  $k$  هو الزمن المتوقع للبحث حتى نهاية اللائحة  $[T[h(k)]]$ ، ذات الطول المتوقع  $E[n_{h(k)}] = \alpha$ . وبذلك، يكون العدد المتوقع للعناصر المفحوصة في بحث غير ناجح هو  $\alpha$ ، والزمن الكلي المطلوب (ومن ضمنه زمن حساب  $h(k)$ ) هو  $\Theta(1 + \alpha)$ . ■

يختلف الوضع قليلًا في حالة البحث الناجح، إذ ليس لكلِّ لائحة الاحتمال نفسه بأن تكون عرضة للبحث. وبدلًا عن ذلك، يتناسب احتمال بحث اللائحة مع عدد العناصر التي تحتويها هذه اللائحة. ومع ذلك، يبقى الزمن المتوقع للبحث  $\Theta(1 + \alpha)$ .

## مراجعة 2.11

يستغرق البحث الناجح في جدول توليد جري فيه تمييز التصاديات باستخدام التشفلة، زمناً في الحالة الوسطى  $\Theta(1 + \alpha)$ ، وذلك بفرض أن التلييد منتظم بسيط.

**البرهان** نفترض أن العنصر الذي نبحث عنه متساوي الاحتمال في أن يكون أيّاً من العناصر  $n$  المخزنة في الجدول. يزيد عدد العناصر المختبرة خلال البحث الناجح عن عنصر  $x$  بمقدار 1 على عدد العناصر التي تظهر قبل  $x$  في لائحة  $x$ . ولما كانت العناصر الجديدة تُوضَع في مقدمة اللائحة، فإن العناصر الواقعة قبل  $x$  في اللائحة تكون قد أدرجت جميعها بعد إدراج العنصر  $x$ . ولمعرفة العدد المتوقع للعناصر المفحوصة، سنأخذ المتوسط على  $n$  عنصراً من نوع  $\equiv$  في الجدول، وهو يساوي 1 مضافاً إليه العدد المتوقع للعناصر المضافة إلى لائحة  $x$  بعد إضافة  $x$  إلى اللائحة. نرمز بـ  $x_i$  إلى العنصر ذي الترتيب  $i$  الذي أدخل إلى الجدول، لكل  $i = 1, 2, \dots, n$ ، ولكن  $k_i = x_i.key$ . نعرّف متحولاً عشوائياً مؤشراً  $X_{ij} = I\{h(k_i) = h(k_j)\}$  لكل  $i, j = 1, 2, \dots, n$ ، وبافتراض أن التلييد منتظم وبسيط، يكون لدينا  $\Pr\{h(k_i) = h(k_j)\} = 1/m$  وباستخدام التوطئة 1.5 يكون لدينا كذلك  $E[X_{ij}] = 1/m$ . وبمما يكون العدد المتوقع للعناصر المفحوصة في البحث الناجح هو

$$\begin{aligned}
 & E\left[\frac{1}{n}\sum_{i=1}^n\left(1 + \sum_{j=i+1}^n X_{ij}\right)\right] \\
 &= \frac{1}{n}\sum_{i=1}^n\left(1 + \sum_{j=i+1}^n E[X_{ij}]\right) \quad (\text{باستخدام خطية التوقع}) \\
 &= \frac{1}{n}\sum_{i=1}^n\left(1 + \sum_{j=i+1}^n \frac{1}{m}\right) \\
 &= 1 + \frac{1}{n \cdot m}\sum_{i=1}^n(n-i) \\
 &= 1 + \frac{1}{n \cdot m}\left(\sum_{i=1}^n n - \sum_{i=1}^n i\right) \\
 &= 1 + \frac{1}{n \cdot m}\left(n^2 - \frac{n(n+1)}{2}\right) \quad (\text{باستخدام المعادلة (1.أ)}) \\
 &= 1 + \frac{n-1}{2m} \\
 &= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}.
 \end{aligned}$$

وبذلك يكون الزمن الكلي المطلوب في حالة البحث الناجح (ومن ضمنه زمن حساب دالة التلييد)

$$\Theta(2 + \alpha/2 - \alpha/2n) = \Theta(1 + \alpha)$$

ماذا يعني هذا التحليل؟ إذا كان عدد شقوب جدول التلييد يتناسب على الأقل مع عدد العناصر في الجدول، يكون لدينا  $n = O(m)$  ومن ثم،  $\alpha = n/m = O(m)/m = O(1)$ . وبذلك، يستغرق البحث وسطيًا زمنًا ثابتًا. ولما كان الإدراج يستغرق زمنًا  $O(1)$  في أسوأ الحالات، ويستغرق الحذف زمنًا  $O(1)$  في أسوأ الحالات حينما تكون اللوائح مضاعفة الربط، فيمكننا أن نفقد جميع العمليات المعجبة في زمن  $O(1)$  وسطيًا.

## لمازبن

### 1-2.11

لنفترض أننا نستخدم دالة تلييد  $h$  لتلييد  $n$  مفتاحًا متمايزًا في صيغة  $T$  طولها  $m$ . وبافتراض أن التلييد منتظمٌ بسيط، ما هو عدد التصادمات المتوقعة؟ وبعبارة أدق، ما هو عدد عناصر المجموعة  $\{ \{k, l\} : k \neq l \text{ and } h(k) = h(l) \}$ ؟

### 2-2.11

وضح ما الذي يحدث عندما تُدرج المفاتيح 5, 28, 19, 15, 20, 33, 12, 17, 10 في جدول تلييد جرى فيه تمييز التصادمات بالسلسلة. افترض أن الجدول يحتوي ■ شقوب، وأن دالة التلييد هي  $h(k) = k \bmod 9$ .

### 3-2.11

يفترض البروفيسور Marcy أنه يمكن الحصول على كسب جوهري في الأداء بواسطة تعديل طريقة السلسلة بحيث نحافظ على كل لائحة مرتبة ترتيبًا مفروغًا. كيف يؤثر تعديل البروفيسور هذا على زمن التنفيذ في حالة البحث الناجح، والبحث غير الناجح، والإدراج، والحذف؟

### 4-2.11

اقترح طريقةً لتحسين تخزين العناصر وإزالة التحصيل ضمن جدول التلييد نفسه، يربط جميع الشقوب غير المستعمدة ضمن لائحة الشقوب الفارغة. افترض أن الشقوب الواحد يمكن أن يُخزن علمًا  $\text{flag}$  إضافةً إلى عنصرٍ ومؤشر أو مؤشرين. يجب أن نفقد جميع العمليات المعجبة وعمليات لائحة الشقوب الفارغة في زمن متوقع  $O(1)$ . هل ينبغي أن تكون لائحة الشقوب الفارغة مضاعفة الترابط، أم يكفي أن تكون بسيطة الترابط؟

### 5-2.11

افترض أننا نُخزّن مجموعةً من  $n$  مفتاحًا في جدول تلييد حجمه  $m$ . نريد أن نعرف إذا كان يجري سحب المفاتيح من مجموعة  $U$  حيث  $|U| > nm$ ، فإن هناك مجموعة جزئية من  $U$  حجمها  $n$  تتكون من المفاتيح التي تلييد

جميعها في الشكيب نفسه، بحيث يكون زمن البحث في أسوأ الحالات في حالة التلييد مع الشكيلة هو  $O(n)$ .

### 6-2.11

افترض أننا نحزنا  $n$  مفتاحاً في جدول تلييد حجمه  $m$ ، نُحلُّ فيه التصادمات بوساطة الشكيلة، وأتأنا نعلم طول كلِّ شكيلة، ومنها الطول  $L$  لأطول شكيلة. اشرح الإجراء الذي يختار عشوائياً مفتاحاً من بين المفاتيح في جدول التلييد وفق توزيع منتظم ويعيده في زمن متوقع  $O(L \cdot (1 + 1/\alpha))$ .

## 3.11 دوال التلييد

سنناقش في هذا الملقط بعض النقاط التي تتعلق بتصميم دوال تلييد جيدة، ثم نعرض ثلاثة مناهج لتعريف هذه الدوال. اتان من هذه المناهج (وهما: التلييد باستخدام التقسيم، والتلييد باستخدام الضرب)، كميَّان heuristic بطبيعتهما، في حين يُستخدم المنهج الثالث (وهو التلييد الشامل)، العشوائية المضافة randomization ويتيح بفضلها أداءً جيداً معزّزاً بالبرهان.

### ما الذي يجعل دالة التلييد جيدة؟

تحقق دالة التلييد الجيدة (تقريباً) افتراض التلييد ذي التوزيع المنتظم البسيط<sup>1</sup>: أي إن لكلِّ مفتاح احتمالاً متساوياً في أن يلبد إلى أيٍّ من الـ  $m$  شكيباً، باستقلالية عن مكان تلييد أي مفتاح آخر. لسوء الحظ، لا يمكننا عادة التحقق من هذا الشرط، لأننا نادراً ما نعرف التوزيع الاحتمالي الذي يجري سحب المفاتيح تبعاً له. يضاف إلى ذلك، أن سحب المفاتيح قد لا يجري باستقلالية.

قد نعرف أحياناً التوزيع. فمثلاً، إذا علمنا أن المفاتيح هي أعداد حقيقية عشوائية  $k$  مُوزَّعة باستقلالية وبانتظام في المجال  $0 \leq k < 1$ ، عندها نحقق دالة التلييد

$$h(k) = [km]$$

شرط التلييد المنتظم البسيط.

عملياً، يمكننا أن نستخدم غالباً تقنيات كسبية لإيجاد دالة تلييد تؤدي أداءً جيداً. وقد تغيد معلومات نوعية عن توزيع المفاتيح في عملية التصميم. لنأخذ، مثلاً، جدول رموز في مترجم compiler مفتاحه متتاليات حرفية تُعَمَّل مُرَفَّقات في برنامج ما. غالباً ما ترد رموز وثيقة العلاقة، مثل  $pt$  و  $pts$ ، في البرنامج نفسه. يُفترض في دالة التلييد الجيدة أن تقلِّل إلى الحد الأدنى فرصة تلييد مثل هذه البدائل في الشكيب نفسه. تُشكِّلُ منهجية جيدة قيمة التلييد بطريقة نأمل أن تكون مستقلة عن أية انحاز قد تكون موجودة في

<sup>1</sup> للتبسيط، سندعو هذا التلييد فيما يلي بالتلييد المنتظم البسيط. (المراجع العلمي)

المعطيات. فمثلاً، نَحْسِبُ "طريقة التقسيم" *division method* (التي نناقشها في المقطع 1.3.11) قيمة التلييد باعتبارها باقي قسمة المفتاح على عددٍ أوليٍّ عَظْمٍ. تعطي هذه الطريقة غالبًا نتائج جيدة، بقرض أننا نختار العدد الأولي بحيث لا يكون متعلقًا بأي من الأنماط في توزيع المفاتيح.

أخيرًا، نلاحظ أن بعض تطبيقات دوال التلييد قد تتطلب خصائص أقوى مما هو متوفر في التلييد المنتظم البسيط. مثلاً، قد نريد أن تعطي مفاتيح "متقاربة" بمعنى ما قيم تلييد متباعدة فيما بينها. (هذه الخاصية مرغوبة خصوصًا عندما نستخدم السير الخطي *linear probing*، المعروف في المقطع 4.1.1) وبوفر التلييد الشامل *universal hashing*، لوصوف في المقطع 3.3.11، الخصائص المرغوبة.

### تفسير المفاتيح كأعداد طبيعية

نفترض معظم دوال التلييد أن المجموعة الشاملة للمفاتيح هي مجموعة الأعداد الطبيعية  $N = \{0, 1, 2, \dots\}$ . فإذا لم تكن المفاتيح أعدادًا طبيعية، نوجد طريقة لتفسيرها كأعداد طبيعية. مثلاً، يمكننا تفسير المتواليات الحرفية كعدد صحيح يُغَيَّرُ عنه بتدوين باستخدام أساس مناسب. وهكذا، فقد نسر الشُعْرَفِ *pt* كزوج من الأعداد العشرية الصحيحة (112, 116)، لأن  $p = 112$  و  $t = 116$  هي مجموعة حروف الـ ASCII، ثم يُعَبَّرُ عن *pt* بوصفه عددًا صحيحًا حسب الأساس-128، فيصبح  $116 + (128 \cdot 112)$ . نقوم عادةً في سياق تطبيقي ما بتصميم طريقة كهذه لتفسير كل مفتاح كعدد طبيعي (قد يكون عددًا كبيرًا). سنفترض فيما يلي أن المفاتيح أعدادٌ طبيعية.

### 1.3.11 طريقة التقسيم

لإنشاء دوال التلييد باستخدام *طريقة التقسيم division method*، نقابل مفتاحًا  $k$  بشَقَرٍ واحد من  $m$  شَقَرًا، وذلك بأخذ باقي قسمة  $k$  على  $m$ . بحيث تكون دالة التلييد هي

$$h(k) = k \bmod m.$$

مثلاً، إذا كان حجم جدول التلييد  $m = 12$  والمفتاح  $k = 100$ ، فإن  $h(k) = 4$ . ولما كانت طريقة التقسيم تحتاج إلى عملية قسمة واحدة، فإن التلييد بالتقسيم سريع جدًا.

حين نستخدم طريقة التقسيم، فإننا نَحْسِبُ عادةً قيمًا معينة لـ  $m$ . مثلاً، يجب ألا تكون  $m$  قوةً من قوى العدد 2، لأنه إذا كانت  $m = 2^p$ ، فستكون الدالة  $h(k)$  مجرد الـ  $p$  بتًا ذات الأدنى مرتبةً من  $k$ . وما لم نكن نعلم أن جميع أنماط الـ  $p$  بتًا الأدنى مرتبةً متساوية الاحتمال، فالأفضل أن نصمّم دالة تلييد تعتمد على جميع بتات المفتاح. يُطْلَبُ إليك في التمرين 3-3.11 أن تُبَيِّنَ أنَّ اختيار  $m = 2^p - 1$ ، قد يكون خيارًا سيئًا، عندما تكون  $k$  متواليات حرفية مفسّرة حسب قوى  $2^p$ ، لأن تبديل ترتيب حروف المفتاح  $k$  لا يغيّر قيمة تلييده. يمثّل عددٌ أوليٌّ غير قريب جدًا من قوة للعدد 2 غالبًا خيارًا جيدًا لقيمة  $m$ . لنفترض مثلاً أننا نرغب

بتخصيص جدول تلييد ثَمَانِيّ التصادمات فيه بالسلسلة، لتخزين  $n = 2000$  متوالية مخفية على وجه التقريب، حيث يمثل الحرف ب 8 بتات. ونحن لا نمانع اعتبار 3 عناصر وسطياً في بحث غير ناجح، لذا نخصص جدول تلييد حجمه  $m = 701$ . اخترنا العدد 701 لأنه عدد أولي قريب من  $2000/3$  لكنّه غير قريب من أي قوة للعدد 2. بمعالجة أي مفتاح  $k$  كعدد صحيح، تكون دالة التلييد

$$h(k) = k \bmod 701 .$$

### 2.3.11 طريقة الضرب

تعمل طريقة الضرب *multiplication method* على إيجاد دوال تلييد على مرحلتين. في المرحلة الأولى نضرب المفتاح  $k$  بثابت  $A$  يقع ضمن المجال  $1 < A < m$  ونستخرج الجزء الكسري من  $kA$ . ثم نضرب هذه القيمة بـ  $m$  ونأخذ أقرب عدد صحيح للنتيجة. وباختصار، دالة التلييد هي

$$h(k) = \lfloor m(kA \bmod 1) \rfloor ,$$

حيث تعني العبارة " $kA \bmod 1$ " الجزء الكسري من  $kA$ ، أي:  $kA - \lfloor kA \rfloor$ .

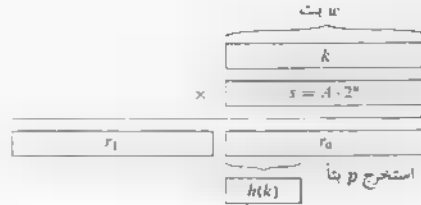
نكمن إحدى حسنات طريقة الضرب في أن قيمة  $m$  ليست ذات أهمية حرجية. ونختارها عادة لتكون قوة من قوى 2 ( $m = 2^p$  و  $p$  عدد صحيح ما) لأننا نستطيع عندها تنجيز الدالة ببساطة على معظم الحواسيب كما يلي. افترض أن حجم كلمة الحاسوب هو  $w$  بتاً، وأن  $k$  يتّسع في كلمة واحدة. نقيّد قيم  $A$  لتكون كسراً من الشكل  $s/2^w$ ، حيث  $s$  عدد صحيح في المجال  $0 < s < 2^w$ . بالإشارة إلى الشكل 4.11، نضرب أولاً  $k$  بالعدد الصحيح ذي الـ  $w$ -بتاً وهو  $s$ ، فتكون النتيجة هي القيمة  $r_0 + r_1 2^w$  التي عرضها  $2w$ -بتاً، حيث  $r_1$  هي كلمة الجداء الأعلى مرتبة، و  $r_0$  هي كلمة الجداء الأدنى مرتبة. وتتألف قيمة التلييد  $p$ -بتاً المطلوبة من  $p$ -بتاً الأكثر دلالة من  $r_0$ .

ومع أن هذه الطريقة تعمل على أيّ قيمة للثابت  $A$ ، إلا أنّها تعمل على نحو أفضل على قيم دون أخرى. ويعتمد الخيار الأمثل على خصائص للمعطيات المُليّدة. يقترح Knuth [211] أنه من المرجّح أن تعمل القيمة

$$A \approx (\sqrt{5} - 1)/2 = 0.6180339887 \dots \quad (2.11)$$

بجودة معقولة.

لنفترض، كمثال على ذلك، أن  $k = 123456$  و  $p = 14$  و  $m = 2^{14} = 16384$  و  $w = 32$ . وبأخذ اقتراح Knuth، نختار  $A$  كسراً من الشكل  $s/2^{32}$  بحيث يكون الأقرب إلى  $(\sqrt{5} - 1)/2$ ، وبحيث  $A \cdot s = 32770602297664 = (2^{32} \cdot 76300) + 17612864$ . إذن  $r_1 = 76300$  و  $r_0 = 17612864$ . وبذلك يكون  $h(k) = 67$  القيمة  $r_0$  بتاً العليا من 14 بتاً الناتج الـ 14 بتاً العليا من  $r_0$ .



**الشكل 4.11** طريقة التليد بالقرب. يجري ضرب تمثيل المفتاح  $k$  بـ  $w = 2^n$  بتًا بالقيمة  $s = A \cdot 2^n$ . تشكل الـ  $p$  بتًا من المرتبة الأعلى من النصف الأدنى ذي العرض  $w$  بتًا من الجداء قيمة التليد المطلوبة  $h(k)$ .

### \* 3.3.11 التليد الشامل

إذا اختار خصم مكر أن تُليد المفاتيح باستخدام دالة تليد ثابتة، عندها يستطيع اختيار  $n$  مفتاحًا تُشكِّلُ جميعها في الشُّبب نفسه، وينتج متوسط زمن استرجاع  $\Theta(n)$ . إن أية دالة تليد ثابتة معرضة لمثل هذا السلوك الرديء في أسوأ الحالات؛ الطريقة الفعالة الوحيدة لتحسين هذا الوضع هي باختيار دالة التليد عشوائيًا بطريقة مستقلة عن المفاتيح التي ستُخزَّنُ فعليًا. يمكن أن تؤدي هذه المنهجية، التي تسمى **التليد الشامل universal hashing**، بالمتوسط إلى أداء جيد يمكن إثباته، بقصع النظر عن المفاتيح التي يختارها الخصم.

في التليد الشامل، نختار في بداية التنفيذ دالة التليد عشوائيًا من مجموعة دوال مصممة. حيث تضمن إضافة العشوائية، كما هو الحال في الفرز السريع، ألا يؤدي دخل وحيد دائمًا إلى سلوك أسوأ الحالات. وبسبب أننا نختار دالة التليد عشوائيًا، تستطيع الخوارزمية أن تسلك سلوكًا مختلفًا في كل تنفيذ، حتى في حالة الدخل نفسه، ضامنة أداءً جيدًا في الحالة الوسطى لأي دخل. وبالعودة إلى مثال جدول رموز المترجم، نجد أن اختيار المربيع المُعرَّفات لا يمكن أن يسبب الآن أداءً تليد سيئًا ثابتًا. ويُحدث الأداء الشئ فقط عندما يختار المترجم دالة تليد عشوائية تؤدي إلى تليد مجموعة من المُعرَّفات برداءة، لكن احتمال حدوث هذا الوضع يبقى صغيرًا، ويُحدث الأمر نفسه لأي مجموعة مُعرَّفات لها الحجم نفسه.

لتكن  $\mathcal{H}$  مجموعة منتهية من دوال التليد التي تقابل علماً معطى من المفاتيح  $U$  بقيم من المجموعة  $\{0, 1, \dots, m-1\}$ . يقال عن مثل هذه المجموعة أنها **شاملة universal** إذا تحقَّق ما يلي: لكل زوج من المفاتيح المتمايزة  $k, l \in U$ ، يكون عدد دوال التليد  $h \in \mathcal{H}$  حيث  $h(k) = h(l)$  هو على الأكثر  $|\mathcal{H}|/m$ . وبعبارة أخرى، باختيار دالة تليد عشوائيًا من  $\mathcal{H}$ ، فإن احتمال التصادم بين مفتاحين متمايزين  $k$  و  $l$  لا يزيد على احتمال  $1/m$  من التصادم إذا اختير  $h(k)$  و  $h(l)$  عشوائيًا وعلى نحو مستقل من المجموعة  $\{0, 1, \dots, m-1\}$ .

تبيّن المبرهنة التالية أن سلوك صف دوال التلييد الشامل سلوكٌ جيد في الحالة الوسطى. نذكر أن  $n_i$  تشير إلى طول اللائحة  $T[i]$ .

### مبرهنة 3.11

افترض أنه يجري اختيار دالة تلييد  $h$  عشوائيًا من مجموعة شاملة من دوال التلييد واستخدمناها لتلييد  $n$  مفتاحًا في جدول  $T$  حجمه  $m$ ، واستخدمنا السلسلة لتمييز التصادم. إذا لم يكن للمفتاح  $k$  في الجدول، عندها يكون الطول المتوقع  $E[n_{h(k)}]$  لللائحة التي يتليد فيها للمفتاح  $k$  هو على الأكثر عامل التحميل  $\alpha = n/m$ . وإذا وُجد المفتاح  $k$  في الجدول، عندها يكون الطول المتوقع  $E[n_{h(k)}]$  لللائحة التي تحتوي المفتاح  $k$  هو  $1 + \alpha$  على الأكثر.

**البرهان** نلاحظ هنا أن التوقعات هي على اختيار دالة التلييد، ولا تعتمد على أية فرضيات تتعلق بتوزيع المفاتيح. نعرف لكل زوج من المفاتيح المتساوية  $k$  و  $l$ ، متحولًا عشوائيًا مؤشرًا  $X_{kl} = I\{h(k) = h(l)\}$ . ولأنه حسب تعريف مجموعة شاملة من دوال التلييد، يتصادم زوج وحيد من المفاتيح باحتمال  $1/m$  على الأكثر، فلدينا  $\Pr[h(k) = h(l)] \leq 1/m$ ، وبذلك تقتضي التوقعة 1.5 أن يكون  $E[X_{kl}] \leq 1/m$ . بعد ذلك، نعرف، لكل مفتاح  $k$ ، المتحول العشوائي  $Y_k$  الذي يساوي عدد المفاتيح غير المفتاح  $k$  التي تتليد في السلسلة نفسه الذي يتليد فيه المفتاح  $k$ ، بحيث

$$Y_k = \sum_{\substack{l \in T \\ l \neq k}} X_{kl}.$$

وبذلك يكون لدينا

$$\begin{aligned} E[Y_k] &= E\left[\sum_{\substack{l \in T \\ l \neq k}} X_{kl}\right] \\ &= \sum_{\substack{l \in T \\ l \neq k}} E[X_{kl}] \quad (\text{باستخدام خطية التوقع}) \\ &\leq \sum_{\substack{l \in T \\ l \neq k}} \frac{1}{m}. \end{aligned}$$

تعتمد تمة البرهان على كون المفتاح  $k$  في الجدول  $T$  أو لا.

• إذا كان  $k \notin T$ ، فإن  $n_{h(k)} = Y_k$  و  $|\{l : l \in T \text{ and } l \neq k\}| = n$ . ومن ثم يكون

$$E[n_{h(k)}] = E[Y_k] \leq n/m = \alpha$$



■ إذا كان  $k \in T$ ، ولما كان المفتاح  $k$  يظهر في اللائحة  $T[h(k)]$  والعدد  $Y_k$  لا يشمل المفتاح  $k$ ، يكون لدينا  $n_{h(k)} = Y_k + 1$  و  $| \{l : l \in T \text{ and } l \neq k\} | = n - 1$ . وهكذا يكون

$$E[n_{h(k)}] = E[Y_k] + 1 \leq (n-1)/m + 1 = 1 + \alpha - 1/m < 1 + \alpha$$

■

النتيجة التالية تقول أن التلييد الشامل يوفر الربح المطلوب: فالآن يصبح من المستحيل أن يجد خصم متتالية من العمليات تُحَوِّر زمن التنفيذ على أن يكون في أسوأ حالاته. وبإضافة عشوائية على اختيار دالة التلييد أثناء التنفيذ، على نحو ذكي، نضمن أن نستطيع معالجة أية متتالية من العمليات، في زمن تنفيذ جيد في الحالة المتوسطة.

#### نتيجة 4.11

باستخدام التلييد الشامل مع تميز التصادم بالمتسلسلة في جدول فارغ في البداية ويحتوي  $m$  شقبة، تستغرق معالجة أية متتالية، مؤلفة من  $n$  عملية INSERT و SEARCH و DELETE وتحتوي  $O(m)$  عملية INSERT، زمنًا متوقعًا  $\Theta(n)$ .

**البرهان** لما كان عدد عمليات الإدراج  $O(m)$ ، فلدينا  $n = O(m)$  وكذلك  $\alpha = O(1)$ . وتستغرق عمليات INSERT و DELETE زمنًا ثابتًا، وحسب لئمة 3.11، فإن الزمن المتوقع لكل عملية SEARCH هو  $O(1)$ . وباستخدام خاصية خطية التوقع ينتج أن الزمن المتوقع لمتتالية العمليات  $n$  كلها هو  $O(n)$ . ولما كانت كل عملية تستغرق زمنًا  $\Omega(1)$ ، فينتج لدينا الحد  $\Theta(n)$ .

■

#### تصميم صف شامل من دوال التلييد

من السهل جدًا تصميم صف شامل من دوال التلييد، لأن نظرية الأعداد الصغيرة تساعدنا على إثبات ذلك. قد تحتاج أولاً للرجوع إلى الفصل 31 إذا لم تكن نظرية الأعداد مألوفة لك.

نبدأ باختيار عدد أولي  $p$  كبير كفاية بحيث يقع كل مفتاح  $k$  في المجال 0 إلى  $p-1$ ، ضمناً. نمرز  $\mathbb{Z}_p$  إلى المجموعة  $\{0, 1, \dots, p-1\}$  و  $\mathbb{Z}_p^*$  إلى المجموعة  $\{1, 2, \dots, p-1\}$ . ولما كان  $p$  عدداً أولياً، فإننا نستطيع حل المعادلات بالمقاس  $p$  باستخدام الطرق للمطاة في الفصل 31. ولأننا نفترض حجم المجموعة الشاملة للمفاتيح أكبر من عدد شقوب جدول التلييد، يكون لدينا  $p > m$ .

نعرف الآن دالة التلييد  $h_{ab}$  لأي  $a \in \mathbb{Z}_p^*$  وأي  $b \in \mathbb{Z}_p$  باستخدام التحويل الخطي متبوعاً باختزالات المقاس  $p$ ، ثم المقاس  $m$ :

$$h_{ab}(k) = ((ak + b) \bmod p) \bmod m. \quad (3.11)$$

فمثلاً، في حال  $p = 17$  و  $m = 6$ ، لدينا  $h_{3,4}(8) = 5$ . وتكون جماعة كل دوال التلييد المماثلة هي

$$\mathcal{H}_{pm} = \{h_{ab} : a \in \mathbb{Z}_p^* \text{ and } b \in \mathbb{Z}_p\}. \quad (4.11)$$

تطابق كل دالة تلييد  $h_{ab}$  المجموعة  $\mathbb{Z}_p$  إلى  $\mathbb{Z}_m$ . ولهذا الصف من دوال التلييد خاصية دقيقة وهي أن حجم مجال الخرج  $m$  اعتباطي — ليس بالضرورة عددًا أوليًا — وهي صمة مستخدمونها في المقطع 5.11. ولما كان لدينا  $p-1$  خيارًا لـ  $a$ ، و  $p$  خيارًا لـ  $b$ ، فالمجموعة  $\mathcal{H}_{pm}$  تحتوي  $p(p-1)$  دالة تلييد.

### مبرهنة 5.11

إن الصف  $\mathcal{H}_{pm}$  من دوال التلييد المعرّف بالمعادلتين (3.11) و (4.11) هو شامل.

**البرهان** نأخذ مفتاحين متميزين  $k$  و  $l$  من  $\mathbb{Z}_p$  حيث  $k \neq l$ . وليكن، في حالة دالة تلييد معطاة  $h_{ab}$

$$r = (ak + b) \bmod p ,$$

$$s = (al + b) \bmod p .$$

نلاحظ أولاً أن  $r \neq s$ . لماذا؟ لاحظ أن

$$r - s \equiv a(k - l) \pmod{p} .$$

وهذا يستتبع أن  $r \neq s$  لأن  $p$  أولي وكلاً من  $a$  و  $(k - l)$  لا يساوي الصفر بالمقاس  $p$ ، إذن يجب أن يكون أحدهما لا يساوي الصفر بالمقاس  $p$  حسب المبرهنة 6.31. لذلك، عند حساب أي  $h_{ab}$  إشارة انتماء  $\mathcal{H}_{pm}$ ، نقابل المدخلات المتميزة  $\blacksquare$  و  $l$  بقيم  $r$  و  $s$  متميزة بالمقاس  $p$  ولن يوجد تصادم على "مستوى المقاس  $p$ ". إضافة إلى ذلك، يؤدي كل من الخيارات  $(p-1)$  للزوج  $(a, b)$  وحيث  $a \neq 0$  إلى زوج ناتج مختلف  $(r, s)$  حيث  $r \neq s$ ، لأننا نستطيع حل  $b$  و  $a$  إذا أعطينا  $r$  و  $s$ :

$$a = ((r - s)((k - l)^{-1} \bmod p)) \bmod p ,$$

$$b = (r - ak) \bmod p ,$$

حيث يشر  $((k - l)^{-1} \bmod p)$  إلى المقلوب الجدالي<sup>2</sup>، الوحيد، بالمقاس  $p$  للمقدار  $(k - l)$ . ولما كان لدينا  $p(p-1)$  زوجاً ممكناً  $(r, s)$  فقط حيث  $r \neq s$ ، فهناك تقابل واحد-إلى-واحد بين الأزواج  $(a, b)$  حيث  $a \neq 0$  والأزواج  $(r, s)$  حيث  $r \neq s$ . وبذلك، وفي حال أي زوج من المدخلات المعطاة  $k$  و  $l$ ، إذا اتفقنا  $(a, b)$  على نحو عشوائي بنظام من  $\mathbb{Z}_p \times \mathbb{Z}_p$ ، فسيكون للزوج الناتج  $(r, s)$  احتمال متساوي في أن يكون أي زوج من القيم المتميزة بالمقاس  $p$ .

ينتج من ذلك أن احتمال تصادم المفتاحين المتميزين  $\blacksquare$  و  $l$  يساوي احتمال  $r \equiv s \pmod{m}$  عند اختيار  $r$  و  $s$  عشوائياً باعتبارهما قيمًا متميزة بالمقاس  $p$ . في حال قيمة معطاة  $r$ ، وللقيم الممكنة المتبقية لـ  $s$ ، وعددها  $(p-1)$ ، فإن عدد قيم  $s$  حيث  $s \equiv r \pmod{m}$  و  $s \neq r$  هي على الأكثر

<sup>2</sup> المقلوب الجدالي بالمقاس  $p$  لعدد ما  $m$  هو العدد الطبيعي الذي يكون ناتج جدائه بـ  $m$  بالمقاس  $p$  هو 1. مثال: 4 هو المقلوب الجدالي بالمقاس 11 للعدد 3 لأن  $4 \times 3 \bmod 11 = 1$ . (للمراجع العلمي)

$$[p/m] - 1 \leq ((p + m - 1)/m) - 1 \quad ((6.3) \text{ حسب المتراجحة})$$

$$= (p - 1)/m .$$

إن احتمال تصادم  $s$  مع  $r$  في حال الاختزال بالمقاس  $m$  هو على الأكثر

$$((p - 1)/m)/(p - 1) = 1/m .$$

وهكذا، في حال أي زوج من القيم المتمايزة  $k, l \in \mathbb{Z}_p$

$$\Pr \{h_{ab}(k) = h_{ab}(l)\} \leq 1/m .$$

■

بمخت يكون  $\mathcal{H}_{pm}$  شاملاً بالفعل.

### تمارين

#### 1-3.11

لتفرض أننا نريد البحث في لائحة مترابطة طويلاً  $m$ ، حيث يحتوي كل عنصر مفتاحاً  $k$  وقيمة تلييد  $h(k)$ . المفتاح  $k$  هو متتالية بحرفية طويلة. كيف يمكننا الاستفادة من قيم التلييد عند البحث في اللائحة عن عنصر له مفتاح معطى؟

#### 2-3.11

افتراض أننا نلبد متتالية من  $r$  محرفاً في  $m$  شُغلاً بمعالجتها كعدد بالأساس-128، ثم باستخدام طريقة التقسيم، نستطيع بسهولة أن نُحُلّ العدد  $m$  ككلمة حاسوب 32-بتاً، فيما تأخذ امتتالية المؤلفّة من  $r$  محرفاً، والمعالج كعدد بالأساس-128، عدة كلمات. كيف يمكننا تطبيق طريقة التقسيم حساب قيمة تلييد متتالية بحرفية دون أن نستخدم أكثر من عدد ثابت من كلمات الذاكرة خارج امتتالية نفسها؟

#### 3-3.11

لتكن لدينا نسخة من طريقة التقسيم فيها دالة التلييد  $h(k) = k \bmod m$ ، حيث  $m = 2^p - 1$  و  $k$  هو متتالية بحرفية تُفسّر في الأساس  $2^p$ . بَيِّنْ أننا لو استطعنا اشتقاق المتتالية  $x$  من المتتالية  $y$  بتبديل محارفها، فإن  $x$  و  $y$  تتلبدان إلى القيمة نفسها. أعط مثلاً من تطبيق تكون فيه هذه الخاصية غير مرغوبة في دالة التلييد.

#### 4-3.11

ليكن لدينا جدول تلييد حجمه  $m = 1000$  ودالة التلييد الموافقة هي  $h(k) = [m(ka \bmod 1)]$  حيث  $A = (\sqrt{5} - 1)/2$ . احسب المواضع التي تتطابق إليها المفاتيح 61 و 62 و 63 و 64 و 65.

#### \* 5-3.11

عرف جماعةً من دوال تلييد  $\mathcal{H}$  من مجموعة متتهية  $U$  في مجموعة متتهية  $B$  لتكون  $\epsilon$ -universal إذا تحقّق لكل الأزواج من العناصر المتمايزة  $k$  و  $l$  في  $U$ ،

$$\Pr\{h(k) = h(l)\} \leq \epsilon .$$

حيث يؤخذ احتمال سحب دالة التلييد  $h$  من الجماعة  $\mathcal{H}$  عشوائيًا. يبين أن عائلة  $\epsilon$ -universal من دوال التلييد يجب أن تحقق

$$\epsilon \geq \frac{1}{|B|} - \frac{1}{|U|}.$$

★ 6-3.11

لتكن  $U$  مجموعة من القيم ذات  $n$  مركبة للسحوبة من  $\mathbb{Z}_p$ ، ولكن  $B = \mathbb{Z}_p$ ، حيث  $p$  عدد أولي. عرف دالة تلييد  $h_b : U \rightarrow B$  حيث  $h_b \equiv b$ ، على دخل ذي  $\equiv$  مركبة  $(a_0, a_1, \dots, a_{n-1})$  من  $U$  بحيث

$$h_b((a_0, a_1, \dots, a_{n-1})) = \left( \sum_{j=0}^{n-1} a_j b^j \right) \bmod p,$$

ولكن  $\mathcal{H} = \{h_b : b \in \mathbb{Z}_p\}$ . ناقش كون  $\mathcal{H}$  هي  $((n-1)/p)$ -شاملة طبقًا لتعريف  $\epsilon$ -universal الوارد في التمرين 5.3.11. (تلميح: انظر التمرين 4.4.11.)

## 4.11 العنوان المفتوحة

في **العنوان المفتوحة open addressing**، تشغل جميع العناصر جدول التلييد نفسه؛ حيث يحتوي كل عنصر لجدول التلييد إما عنصرًا من المجموعة الديناميكية واما NIL. في حال البحث عن عنصر ما، نفحص شقوب الجدول بانتظام حتى نجد العنصر المطلوب أو نتحقق من عدم وجود العنصر في الجدول. وعلى عكس السلسلة، لا توجد لوائح ولا عناصر مخزنة خارج الجدول. وهكذا، في العنوان المفتوحة، يمكن أن "يمتلئ" جدول التلييد بحيث لا يمكن تنفيذ عمليات إدراج إضافية؛ وبالنتيجة لا يمكن أن يتجاوز معامل التحميل  $\alpha$  القيمة 1.

لا شك في أنه كان باستطاعتنا تخزين لوائح مترابطة - لسلسلةها داخل الجدول - في شقوب الجدول غير المستخدمة (انظر التمرين 4.2.11)، غير أن الفائدة من العنوان المفتوحة هي أنها تتجنب المؤشرات تمامًا. وعوضًا عن تتبع المؤشرات، نحسب تنالي الشقوب الواجب فحصها. وتوفر الذاكرة الإضافية المحزنة بسبب عدم تخزين المؤشرات عددًا أكبر من الشقوب، لحجم الذاكرة نفسه، مؤدبة إلى تصادمات أقل واسترجاع أسرع.

لتنجز الإدراج باستخدام العنوان المفتوحة، نفحص على التتابع، أو نسبر *probe*، جدول التلييد حتى نجد شقبةً فارغةً نضع فيه المفتاح. وعوضًا عن أن نكون مقيدين بالترتيب  $0, 1, \dots, m-1$  (الذي يحتاج إلى زمن بحث  $\Theta(n)$ )، تعتمد متتالية المواقع التي تُستَبَر على المفتاح الذي نُدرجه. ولتحديد الشقوب التي ينبغي سيرها، نوسّع دالة التلييد لتشمل عدد السير (ابتداءً من 0) باعتباره دخلًا ثانيًا. وهكذا، تصبح دالة التلييد

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}.$$

في العنونة المفتوحة، نطلب لكل مفتاح  $k$ ، أن تكون متتالية السبر *probe sequence*

$$(h(k, 0), h(k, 1), \dots, h(k, m-1))$$

تبدلياً من  $\{0, 1, \dots, m-1\}$ ، بحيث يُعتبر كل موقع من جدول التليد شُعباً لمفتاح جديد فيما يتعلق الجدول. سنفترض في شبه الرماز التالي أن العناصر في جدول التليد  $T$  هي مفاتيح بدون معلومات تابعة؛ وأن المفتاح  $k$  متطابق مع العنصر الذي يحتوي للمفتاح  $k$ . إن كل شُعبٍ يحوي إما مفتاحاً أو NIL (إذا كان الشُعب فارغاً). وبأخذ الإجراء HASH-INSERT في مدخلاته جدول تليد  $T$  ومفتاحاً  $k$ . وببعد رقم الشُعب حيث يُخزّن المفتاح  $k$  أو أن يُظهر خطأ لأن جدول التليد ممتلئ. حالتي.

HASH-INSERT( $T, k$ )

```

1   $i = 0$ 
2  repeat
3       $j = h(k, i)$ 
4      if  $T[j] == \text{NIL}$ 
5           $T[j] = k$ 
6          return  $j$ 
7      else  $i = i + 1$ 
8  until  $i == m$ 
9  error "hash table overflow"
```

تسبر خوارزمية البحث عن مفتاح  $k$  متتالية الشُعب بنفسها التي فحصتها خوارزمية الإدراج عند إدراج المفتاح  $k$ . لذا يمكن أن يتوقف البحث (محظناً) عندما يجد شُعباً فارغاً، لأن  $k$  كان سيُدرج في هذا الشُعب وليس بعده حسب متتالية سبره. (نفترض هذا النقاش أن المفاتيح لا تُحذف من جدول التليد). بأخذ الإجراء HASH-SEARCH في مدخلاته جدول التليد  $T$  ومفتاحاً  $k$ ، ويعيد  $j$  إذا كان الشُعب  $j$  يحتوي للمفتاح  $k$ ، أو NIL إذا لم يكن المفتاح  $k$  موجوداً في الجدول  $T$ .

HASH-SEARCH( $T, k$ )

```

1   $i = 0$ 
2  repeat
3       $j = h(k, i)$ 
4      if  $T[j] == k$ 
5          return  $j$ 
6       $i = i + 1$ 
7  until  $T[j] == \text{NIL}$  or  $i == m$ 
8  return NIL
```

إن الحذف من جدول تليد بعنونة مفتوحة صعب. فعندما نحذف مفتاحاً  $k$  من شُعب  $i$ ، لا نستطيع ببساطة أن نُعلم ذلك الشُعب على أنه شُعب فارغ بأن نُخزّن فيه NIL. لأننا إذا فعلنا ذلك فقد يكون من المستحيل استعادة أي مفتاح  $k$  كنا قد سبرنا خلال إدراجه الشُعب  $i$  ووجدناه مشغولاً. نستطيع أن نحل هذه

المسألة بأن نجعل الشقب يُخزَّن القيمة الخاصة DELETED عوضاً عن القيمة NIL. في هذه الحالة، علينا تعديل الإجراء HASH-INSERT لمعالجة هذا الشقب كما لو أنه كان فارغاً بحيث نستطيع أن نُدرج فيه مفتاحاً جديداً. أما الإجراء HASH-SEARCH فلا نحتاج إلى تعديله، لأنه سيتخطى قيم DELETED أثناء البحث. ولكن، حين نستخدم القيمة الخاصة DELETED، تصبح أزمنة البحث غير معتمدة على معامل التحميل  $\alpha$ ، ولهذا السبب يجري عموماً اختيار الثُلثيَّة بصفتها نقانة لتمييز التصادم حين يجب حذف مفاتيح.

نفترض في تحليلنا أن التلييد منتظم *uniform hashing*: وهذا يعني أن متتالية السر لأي مفتاح لها الاحتمال نفسه في أن تأخذ أيّاً من  $m$  تبديلاً من  $\{0, 1, \dots, m-1\}$ . يُفكَّم التلييد المنتظم فكرة التلييد المنتظم البسيط المعرّف سابقاً إلى الحالة التي تنتج فيها دالة التلييد ليس فقط عدداً مفرداً، بل متتالية سر كاملة. إن التلييد المنتظم الحقيقي صعب التحقيق، إلا أننا نستخدم عملياً تقريبات مناسبة (مثل التلييد المضاعف، المُعرّف لاحقاً).

سنختار ثلاث طرق شائعة الاستخدام لحساب متاليات السر المطلوبة في العنونة المفتوحة: السر الخطي *linear probing* والسر التربيعي *quadratic probing* والتلييد للمضاعف *double hashing*. تضمن هذه التقانات أن يكون  $\{h(k, 0), h(k, 1), \dots, h(k, m-1)\}$  تبديلاً من  $\{0, 1, \dots, m-1\}$  لكل مفتاح  $k$ . ولكن، لا تلي أيّ من هذه التقانات فرضية التلييد المنتظم، لأن أيّاً منها غير قادر على توليد أكثر من  $m^2$  متتالية سر مختلفة (عوضاً عن  $m!$  التي يتطلبها التلييد المنتظم). ينجز التلييد المضاعف أكبر عدد من متاليات السر، وكما قد يتوقع المرء، يبدو أنه يعطي أفضل النتائج.

### السر الخطي

لنكن لدينا دالة تلييد عادية  $h' : U \rightarrow \{0, 1, \dots, m-1\}$ ، نسميها *دالة تلييد مساعدة auxiliary hash function*، نستخدم طريقة *السر الخطي linear probing* دالة التلييد

$$h(k, i) = (h'(k) + i) \bmod m$$

حيث  $i = 0, 1, \dots, m-1$ . ليكن لدينا المفتاح  $k$ ، نسر أولاً الشقب  $T[h'(k)]$ ، أي الشقب للمعطى بواسطة دالة التلييد المساعدة. ثم نسر الشقب  $T[h'(k) + 1]$ ، وهكذا حتى الشقب  $T[m-1]$ . ثم نلف عائدين إلى الشقوب  $T[0], T[1], \dots$  حتى نسر أخيراً الشقب  $T[h'(k) - 1]$ . ولما كان الشقب الذي يبدأ عنده السر هو الذي يحدّد كامل متتالية السر، فيوجد  $m$  متتالية سر متمايزة فقط.

إن السر الخطي سهل التحيز، لكنه يعاني من مشكلة تُعرّف *بالعنقدة الأولية primary clustering*. إذ تنشأ تدريجيّاً سلاسل طويلة بسبب الشقوب المشغولة، تؤدي إلى زيادة زمن البحث الوسطي. وتظهر العناقيد لأن الشقب الفارغ للسبوق بـ  $i$  شقياً عتلاً سيّئاً بعدها باحتمال  $(i+1)/m$ . تتزعج السلاسل الطويلة بسبب الشقوب المشغولة لتصبح أطول، ويزداد بذلك زمن البحث الوسطي.

### المسار التريبيعي

يستخدم المسار التريبيعي *quadratic probing* دالة تلييد من الشكل

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m, \quad (11.5)$$

حيث  $h'$  دالة تلييد مساعدة، و  $c_1$  و  $c_2$  ثابتان مساعدتان موجبان، و  $i = 0, 1, \dots, m-1$ . إن أول موقع سيحري سره هو  $T[h'(k)]$  والمواقع التالية التي سيتم سرها هي إزاحات بقيم تعتمد بطريقة تربيعية على رقم المسار  $i$ . إن أداء هذه الطريقة أفضل كثيراً من المسار الخطي، لكن حتى نستخدم كامل جدول التلييد، نُقيّد قيم  $c_1$  و  $c_2$  و  $m$ . توضّح المسألة 11-3 طريقة لاختيار هذه المتوسطات. وكذلك، إذا كان لمفتاحين موضع المسار البدائي نفسه، يكون لما متتالية السر نفسها، لأن  $h(k_1, 0) = h(k_2, 0)$  ينطوي ضمنياً على  $h(k_1, i) = h(k_2, i)$ . تؤدي هذه الخاصية إلى شكل متزايد من العقدة، يسمى *العقدة الثانوية secondary clustering*. وكما في المسار الخطي، يُعدّد المسار البدائي كامل التتالية. وهكذا يُستخدم  $m$  متتالية متمايزة فقط.

### التلييد المضاعف

يتيح التلييد المضاعف إحدى أفضل الطرائق للناحية للعتونة المفتوحة لأن التبادل الناتجة تحتل كثيراً من خواص *التباديل permutations* المختارة عشوائياً. يُستخدم *التلييد المضاعف double hashing* دالة تلييد من الصيغة

$$h(k, i) = (h_1(k) + i h_2(k)) \bmod m,$$

حيث  $h_1$  و  $h_2$  دالتا تلييد مساعدتان. يبدأ السر في الموقع  $T[h_1(k)]$  وتكون مواقع المسار المتعاقبة هي إزاحة عن المواقع السابقة بمقدار القيمة  $h_2(k)$  بالمقاس  $m$ . ومن ثم، وخلافاً لحالة المسار الخطي والتريبيعي، يعتمد تنامي السر هنا من جانبيه على المفتاح  $k$ ، لأنه قد يتغير موقع السر الابتدائي، أو الإزاحة، أو كلاهما. يعطي الشكل 11.1 مثالاً على الإدراج باستخدام التلييد المضاعف.

يجب أن تكون القيمة  $h_2(k)$  وحجم جدول التلييد بأكمله  $m$  أوليين فيما بينهما حتى يكون بالإمكان البحث في كامل جدول التلييد. (انظر التمرين 11.4-4) وتكمن إحدى الطرق المناسبة لضمان هذا الشرط في جعل  $m$  قوة من قوى العدد 2 ولتصميم  $h_2$  بحيث تُنتج عدداً فردياً دوماً. ثمة طريقة أخرى نجعل فيها  $m$  عدداً أولياً ونصمم  $h_2$  بحيث تعيد دوماً عدداً صحيحاً موجباً أقل من  $m$ . فمثلاً، يمكن أن نختار  $m$  عدداً أولياً ونجعل

$$h_1(k) = k \bmod m,$$

$$h_2(k) = 1 + (k \bmod m'),$$

حيث نختار  $m'$  أقل قليلاً من  $m$  (وليكن  $m-1$ ). مثلاً، إذا كان  $k = 123456$  و  $m = 701$

0	
1	79
2	
3	
4	69
5	98
6	
7	72
8	
9	14
10	
11	50
12	

**الشكل 5.11** الإدراج باستخدام التلييد المضاعف. لدينا هنا جدول تلييد حجمه 13 و  $h_1(k) = k \bmod 13$  و  $h_2(k) = 1 + (k \bmod 11)$ . ولما كان  $14 \equiv 1 \pmod{13}$  و  $14 \equiv 3 \pmod{11}$ ، فإننا نُدرج للفتاح 14 في الشُّقْب الفارغ 9، بعد أن نُفحص الشُّقْبَيْن 1 و 5 ونجد أنهما مشغولان.

و  $m' = 700$ ، فلدينا  $h_1(k) = 80$  و  $h_2(k) = 257$ ، وبذلك نَسْتَبْرُ أولاً للوقع 80، ثم نفحص كل 257 شقْب (بالمقاس  $m$ ) إلى أن نجد للفتاح أو نُفَصِّصَ كل الشُّقُوب.

حينما يكون  $\equiv$  عدداً أولياً أو من قوى العدد 2، يقدم التلييد المضاعفُ تحسينات مقارنةً بالسِر الخطي والسِر التربيعي بأنه يَسْتَحْدِثُ متتاليات عددها  $\Theta(m^2)$ ، عوضاً عن  $\Theta(m)$ ، لأن كل زوج مُخْتَمَل  $\{h_1(k), h_2(k)\}$  يؤدي إلى متتالية سِر متميزة. يظهر إذن أن أداة التلييد المضاعف، مثل هذه القيم من  $m$ ، قريب جداً من أداء السلوك "المثالي" للتلييد المنتظم.

بالرغم من أنه يمكن من حيث المبدأ اختيار  $m$  مختلفة عن الأعداد الأولية وقوى العدد 2 لاستخدامها في التلييد المضاعف، إلا أن ذلك يجعل إيجاد طريقة فعالة لتوليد  $h_2(k)$  بحيث يكون أولاً مع  $m$  أكثر صعوبة. وأحد أسباب ذلك هو ضعف الكثافة النسبية لكل هذه الأعداد الأولية مع  $m$ ، المقدَّر بـ  $\phi(m)/m$  (انظر المعادلة 24.31).

### تحليل تلييد العنونة المفتوحة

نُعَبِّرُ عن تحليلنا للعنونة المفتوحة، كما في تحليلنا للسلسلة، بدلالة معامل تحميل جدول التلييد  $\alpha = n/m$ . طبعاً، في العنونة المفتوحة، يشغل كلُّ شقْبٍ عنصرٌ واحدٌ على الأكثر، ومن ثم فإن  $n \leq m$ ، وهذا يقتضي أن  $\alpha \leq 1$ .

نفترض أننا نستخدم التلييد المنتظم. وتبعاً لهذه الفرضية المثالية، تكون متتالية السِر  $\{h(k, 0), h(k, 1), \dots, h(k, m-1)\}$  المستخدمة لإدراج أي مفتاح  $k$  أو البحث عنه متساوية الاحتمال في



أن تكون أيًا من التباديل  $\{0, 1, \dots, m-1\}$ . بالطبع، لكل مفتاح معطى متالية سر ثابتة وحيدة مرتبطة به؛ ما نقصده هنا، هو أنه بالأخذ بالحسبان توزيع الاحتمال على فضاء المفاتيح وتطبيق دالة التلييد على المفاتيح، تكون أية متالية سر ممكنة متساوية الاحتمال.

نحل الآن عدد مرات السر للتوقعة لتلييد باستخدام العنونة المفتوحة بفرض أن التلييد منتظم، متبدلين بتحليل عدد حالات السر المنفذة في حالة بحث غير ناجح.

### مبرهنة 6.11

ليكن لدينا جدول تلييد ذو عنونة مفتوحة شُعائل' تحميلة  $\alpha = n/m < 1$ ، عندها يساوي العدد المتوقع من السُور [جمع سُور] في بحث غير ناجح  $1/(1-\alpha)$  على الأكثر، وذلك بافتراض أن التلييد منتظم.

**البرهان** إن كل سر - في بحث غير ناجح - ما عدا السر الأخير يُنقُذ إلى شُقب مشغول لا يحتوي المفتاح المطلوب، والشُقب الأخير للبحث فارغ. نعرّف المتحول العشوائي  $X$  على أنه عدد السور المنقُذة في بحث غير ناجح، ونعرّف الحدث  $A_i$ ، حيث  $i = 1, 2, \dots$ ، على أنه الحدث المتمثل في حدوث السر ذي الرقم  $i$ ، والذي يسر شُقباً مشغولاً. فيكون الحدث  $\{X \geq i\}$  هو تقاطع الأحداث  $A_1 \cap A_2 \cap \dots \cap A_{i-1}$ . سنُخذ  $\Pr\{X \geq i\}$  بحذ المقدار  $\Pr\{A_1 \cap A_2 \cap \dots \cap A_{i-1}\}$ . وباستخدام التمرين 5-2،

$$\Pr\{A_1 \cap A_2 \cap \dots \cap A_{i-1}\} = \Pr\{A_1\} \cdot \Pr\{A_2|A_1\} \cdot \Pr\{A_3|A_1 \cap A_2\} \cdots \Pr\{A_{i-1}|A_1 \cap A_2 \cap \dots \cap A_{i-2}\}.$$

ولما كان لدينا  $n$  عنصراً و  $m$  شُقباً، فإن  $\Pr\{A_1\} = n/m$ . في حالة  $i > 1$ ، فإن احتمال وجود سر ذي ترتيب  $i$  لشُقب مشغول هو  $(n-j+1)/(m-j+1)$ ، وذلك بافتراض أن أول  $j-1$  سراً كانت لشُقوب مشغولة. ينتج هذا الاحتمال لأننا قد نجد عنصراً من الـ  $(n-(j-1))$  عنصراً المتبقية، في شُقب من الـ  $(m-(j-1))$  شُقباً غير المفحوصة، وبافتراض أن التلييد منتظم، يكون الاحتمال هو نسبة هاتين الكميتين. وبملاحظة أن  $n < m$  يقتضي  $n/(m-j) \leq (n-j)/(m-j) \leq n/m$  لكل  $j$  بحيث  $0 \leq j < m$ . لدينا لكل  $i$  حيث  $1 \leq i \leq m$ ،

$$\begin{aligned} \Pr\{X \geq i\} &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\ &\leq \left(\frac{n}{m}\right)^{i-1} \\ &= (\alpha)^{i-1}. \end{aligned}$$

الآن نستخدم للمعادلة (ت.25) لحذ العدد المتوقع من السور:

$$E[X] = \sum_{i=1}^{\infty} \Pr\{X \geq i\}$$

$$\begin{aligned} &\leq \sum_{i=1}^{\infty} \alpha^{i-1} \\ &= \sum_{i=0}^{\infty} \alpha^i \\ &= \frac{1}{1-\alpha}. \end{aligned}$$

هذا الحد من  $1/(1-\alpha) = 1 + \alpha + \alpha^2 + \alpha^3 + \dots$  تفسير بديهي: نُنفذ دائماً أول سير، وباحتمال تقريبي  $\alpha$ ، نجد السير الأول شغلاً مشغولاً بحيث نحتاج إلى إجراء سير ثانٍ. وباحتمال تقريبي  $\alpha^2$ ، يكون الشغبان الأولان مشغولين بحيث نجرى سيراً ثالثاً، وهكذا.

إذا كان  $\alpha$  ثابتاً، فإن المبرهنة 6.11 تنبأ بأن بحثاً غير ناجح ينقُذ في زمن  $O(1)$ . فمثلاً إذا كان جدول التلييد نصف مثلي، فالعدد المتوسط للسيرور في بحث غير ناجح هو على الأكثر  $2 = 1/(1-0.5)$ . أما إذا كان ممتلئاً بنسبة 90 بالمئة، فيكون العدد المتوسط للسيرور على الأكثر  $10 = 1/(1-0.9)$ .

تعطينا المبرهنة 6.11 أداء الإجراء HASH-INSERT مباشرةً تقريباً.

### نتيجة 7.11

يتطلب إدراج عنصر في جدول تلييد عنقوة مفتوحة معامل تحميله  $\alpha$ ،  $1/(1-\alpha)$  سيراً وسطياً على الأكثر، وذلك بافتراض أن التلييد منتظم.

**البرهان** ندرج عنصراً في حال توافر فراغ في الجدول فقط، وبذلك يكون  $\alpha < 1$ . ويتطلب إدراج مفتاح بحثاً غير ناجح يتبعه وضع المفتاح في أول شغل بحده فارغاً. ومن ثَمَّ، يكون العدد المتوقع للسيرور  $1/(1-\alpha)$  على الأكثر.

علينا العمل أكثر قليلاً لحساب عدد السيرور المتوقع في حالة بحث ناجح.

### مبرهنة 8.11

إذا كان لدينا جدول تلييد عنقوة مفتوحة معامل تحميله  $\alpha < 1$ ، فإن عدد السيرور المتوقع في البحث الناجح هو على الأكثر

$$\frac{1}{\alpha} \ln \frac{1}{1-\alpha},$$

وذلك بافتراض أن التلييد منتظم، وأن احتمالات البحث عن أي مفتاح في الجدول متساوية.

**البرهان** يُعتمد البحث عن مفتاح ما  $k$  إتناج متتالية السير نفسها، التي أُنشئت أثناء إدراج العنصر ذي المفتاح  $k$ . وحسب النتيجة 7.11، إذا كان  $k$  هو المفتاح ذو الترتيب  $(i+1)$  الذي أدرج في الجدول، فعدد السيرور

المتوقع للبحث عن المفتاح  $k$  هو على الأكثر  $m/(m-i) = 1/(1-i/m)$ . إن حساب متوسط كل المفاتيح  $n$  في جدول التليد يعطينا العدد المتوقع للسيور في حالة بحث ناجح:

$$\begin{aligned} \frac{1}{n} \sum_{i=0}^{n-1} \frac{m}{m-i} &= \frac{m}{n} \sum_{i=0}^{n-1} \frac{1}{m-i} \\ &= \frac{1}{\alpha} \sum_{k=m-n+1}^m \frac{1}{k} \\ &\leq \frac{1}{\alpha} \int_{m-n}^m (1/x) dx \quad (\text{باستخدام المتراجحة (أ. 12)}) \\ &= \frac{1}{\alpha} \ln \frac{m}{m-n} \\ &= \frac{1}{\alpha} \ln \frac{1}{1-\alpha} . \end{aligned}$$

إذا كان جدول التليد نصف ممتلئ، يكون عدد السيور المتوقع في بحث ناجح أقل من 1.387. وإذا كان جدول التليد ممتلئًا بنسبة 90 بالمائة، يكون عدد السيور المتوقع أقل من 2.559.

## تمارين

### 1-4.11

ادرس إدراج المفاتيح 10, 22, 31, 4, 15, 28, 17, 88, 59 في جدول تليد طوله 11 ■ باستخدام العنونة المفتوحة، حيث دالة التليد المساعدة  $h'(k) = k$ . وضح نتيجة إدراج هذه المفاتيح باستخدام السر الخطي، وباستخدام السر التربيعي حيث  $c_1 = 1$  و  $c_2 = 3$ . وباستخدام التليد المضاعف حيث  $h_1(k) = k$  و  $h_2(k) = 1 + (k \bmod (m-1))$ .

### 2-4.11

اكتب شبه رماز للإجراء HASH-DELETE كاتلغص في النص، وعدّل الإجراء HASH-INSERT كي يعالج القيمة الخاصة DELETED.

### 3-4.11

ادرس جدول تليد بعنونة مفتوحة حيث التليد منتظم. أعط حدودًا عليا لعدد السيور المتوقع في بحث غير ناجح، ولعدد السيور المتوقع في بحث ناجح حين يكون معامل التحميل  $3/4$ ، ثم حين يكون  $7/8$ .

### \* 4-4.11

افترض أننا نستخدم التليد المضاعف لتمييز التصادفات - أي نستخدم دالة التليد  $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$ . يبيّن أنه إذا كان  $d \geq 1$  هو القاسم المشترك الأعظم لكل من  $m$  و  $h_2(k)$  في حالة مفتاح ما  $k$ ، فإن بمقا غير ناجح عن المفتاح  $k$  يتكرر حتى الشقّب ذي الترتيب  $1/d$  من

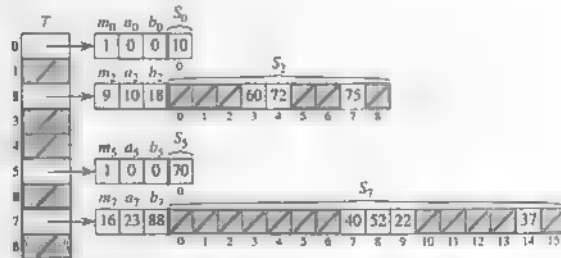
جدول التلييد قبل أن يعود إلى الشقب  $h_1(k)$ . وأنه إذا كان  $d = 1$  و  $m$  و  $h_2(k)$  أُؤثَّرتين فيما بينهما، فإن البحث قد يتحصى كامل جدول التلييد. (تلميح: انظر الفصل 3.1).

#### \* 5-4.11

ادرس جدول تلييد بعنوان مفتوحة معامل تحميله  $\alpha$ . أوجد القيمة  $\alpha$  المغايرة للصفر التي يكون عندها العدد المتوقع للسيور لبحث غير ناجح مساوياً لضعف العدد المتوقع للسيور في بحث ناجح. استخدم الحدود العليا المعطاة في المبرهتين 6.11 و 8.11 لهذه الأعداد المتوقعة للسيور.

### 5.11 ■ التلييد الكامل

إضافة إلى أن التلييد هو على الأغلب خيار جيد من أجل أدائه الرائع في المتوسط، فيمكن أن يوفر أيضاً أداءة رائعة في أسوأ الحالات عندما تكون مجموعة المفاتيح ساكنة *static*: وذلك لأنه بمجرد تخزين المفاتيح في الجدول، فإن مجموعة المفاتيح لا تتغير أبداً. وثمة تطبيقات لها، بصورة طبيعية، مجموعات مفاتيح ساكنة: كمجموعة الكلمات المحفوظة في لغة برمجة، أو الأسماء على قرص مراص CD-ROM. نسمي تقنية التلييد *تلييداً كاملاً* *perfect hashing* إذا كان عدد مرات النفاذ إلى الذاكرة المطلوبة لتنجز بحث في أسوأ الحالات هو  $O(1)$ . لإيجاد أسلوب تلييد كامل، نستخدم تلييداً ذا مستويين، كلٌ منهما تلييد شامل. يوضح الشكل 6.11 هذا النهج.



الشكل 6.11 استخدام التلييد الكامل لتخزين المجموعة  $K = \{10, 22, 37, 40, 52, 60, 70, 72, 75\}$ . دالة التلييد الخارجية هي  $h(k) = ((ak + b) \bmod p) \bmod m$  حيث  $a = 3$  و  $b = 42$  و  $p = 101$  و  $m = 9$ . مثلاً،  $h(75) = 2$ ، وهكذا يتليد المفتاح 75 في الشقب 2 للجدول  $T$ . يُخزَّن جدول تلييد ثانوي  $S_2$  جميع المفاتيح التي تتليد في الشقب  $i$ . حجم الجدول  $S_i$  هو  $m_i = n_i^2$  ودالة التلييد المرافقة هي  $h_i(k) = ((a_i k + b_i) \bmod p) \bmod m_i$ . ولما كان  $h_2(75) = 7$ ، فإن المفتاح 75 يُخزَّن في الشقب 7 للجدول التلييد الثانوي  $S_2$ . لا تحدث أية تصادمات في أيٍّ من جداول التلييد الثانوية. وبذلك، يستغرق البحث زمناً ثابتاً في أسوأ الحالات.

أما المستوى الأول فهو نفسه التلييد مع السلسلة جوهريًا: نُكَبِّد الـ  $n$  مفتاحًا في  $m$  شُكْبًا باستخدام دالة تلييد  $h$  نختارها بعناية من جماعة دوال تلييد شاملة.

ولكن، بدلاً من تكوين لائحة من المفاتيح تليد في الشقب  $i$ ، نستخدم **جدول تلييد ثانوي**  $S_j$  *secondary hash table* دالة تلييد للمرافقة هي  $h_j$ . وباختيار دوال التلييد  $h_j$  بعناية، نضمن عدم وجود تصادمات في المستوى الثاني.

ولكي نضمن عدم وجود تصادمات في المستوى الثاني، نحتاج أن نجعل  $m_j$  وهو حجم جدول التلييد  $S_j$ ، يساوي مربع العدد  $n_j$  الذي هو عدد المفاتيح التي تليد في الشقب  $j$ . وقد تظن أن الاعتماد التريخي لـ  $m_j$  على  $n_j$  قد يبدو أنه يؤدي على الأرجح إلى أن يصبح متطلب التخزين الكلي مفرطًا في الكبر، غير أننا سنبين أنه باختيار دالة تلييد المستوى الأول بعناية، يمكننا الحد من المقدار الكلي المتوقع لفضاء الذاكرة المستخدم إلى رتبة  $O(n)$ .

نستخدم دوال تلييد اختبرت من صفوف شاملة من دوال التلييد المذكورة في المقطع 3-3.11. تأتي دالة تلييد المستوى الأول من الصف  $\mathcal{H}_{pm}$ ، حيث - كما في المقطع 3-3.11 -  $p$  عدد أولي أكبر من قيمة أي مفتاح. يعاد تلييد هذه المفاتيح المثلثة أصلاً في الشقب  $j$  في جدول تلييد ثانوي  $S_j$  بحجمه  $m_j$  باستخدام دالة التلييد  $h_j$  التي نختارها من الصف  $\mathcal{H}_{p,m_j}$ <sup>3</sup>.  
ستابع العمل على مرحلتين. أولاً، نحدد كيفية التحقق من أن الجداول الثانوية لا تحتوي أي تصادمات. ثانياً، نبين أن الكمية المتوقعة من الذاكرة المستخدمة إجمالاً - لجدول التلييد الأولي وجميع جداول التلييد الثانوية - هي  $O(n)$ .

### مبرهنة 9.11

افترض أننا خزننا  $n$  مفتاحًا في جدول تلييد حجمه  $m = n^2$  باستخدام دالة تلييد  $h$  مختارة عشوائيًا من صف شامل من دوال تلييد، عندها يكون احتمال وجود أي تصادمات أقل من  $1/2$ .

**البرهان** يوجد  $\binom{n}{2}$  زوجًا من المفاتيح التي يمكن أن تصادم؛ وكل زوج يمكن أن يتصادم باحتمال  $1/m$  إذا اختبرت  $h$  عشوائيًا من جماعة شاملة  $\mathcal{H}$  من دوال التلييد. ليكن  $X$  متحولًا عشوائيًا يقدّر عدد التصادمات. عندما  $m = n^2$ ، يكون العدد المتوقع للتصادمات

$$E[X] = \binom{n}{2} \cdot \frac{1}{n^2}$$

<sup>2</sup> في حال  $n_j = m_j = 1$ ، فإننا لا نحتاج حقيقة إلى دالة تلييد للشقب  $j$ ؛ فعندما نختار دالة تلييد  $h_{ab}(k) = ((ak + b) \bmod p) \bmod m_j$  لهذا الشقب، نجعل  $a = b = 0$  فقط.

$$= \frac{\pi^2 - \pi}{2} \cdot \frac{1}{\pi^2}$$

$$< \frac{1}{2}.$$

(إن هذا التحليل شبيه بتحليل متناقضة عيد الميلاد في المقطع 4.5-1). وتطبيق مترابطة ماركوف (ت.30)،

$$Pr\{X \geq t\} \leq E[X]/t, \text{ حيث } t = 1, \text{ يكتمل البرهان.}$$

في الحالة التي جرى وصفها في المبرهنة 9.11، حيث  $m = n^2$ ، ينتج أن الاحتمال الأكبر لأن تكون دالة التلييد  $h$  التي جرى اختيارها من  $\mathcal{H}$  عشوائيًا خالية من التصادمات. لتكن لدينا المجموعة  $K$  المؤلفة من  $n$  مفتاحًا (تذكر أن  $K$  ساكنة)، إن من السهل إيجاد دالة تلييد  $h$  خالية من التصادمات بتجارب عشوائية قليلة. ولكن، في حالة  $n$  كبيرة، يكون جدول التلييد ذو الحجم  $\pi^2 = m$  مفرطًا في الكبر. لذا، نستخدم منهج التلييد الثنائي المستوى، ونستخدم منهج المبرهنة 9.11 فقط لتلييد العناصر داخل كل شُئْب. ونستخدم دالة تلييد  $h$  خارجية، أو ذات مستوى أول، لتلييد المفاتيح في  $\pi = m$  شُئْبًا. عندها، إذا تليد  $n_j$  مفتاحًا في شُئْب  $z$ ، نستخدم جدول تلييد ثانوي  $R_z$  حجمه  $m_j = \pi_j^2$  لتوفير بحث ذي زمن ثابت خالي من التصادم. نعود الآن إلى قضية التحقق من أن الذاكرة الإجمالية المستخدمة هي  $O(n)$ . لما كان حجم جدول التلييد الثانوي ذي الترتيب  $z$  ينمو تربيعيًا مع عدد المفاتيح المعززة  $m_j$  فإننا نجازف في أن يكون حجم الذاكرة الإجمالية مفرطًا في الكبر.

إذا كان حجم جدول المستوى الأول  $m = n$ ، فإن حجم الذاكرة المستخدمة في حالة جدول التلييد الأولي يكون  $O(n)$ ، وذلك لحزن الحجم  $m_j$  لجدول التلييد الثانوي، ولحزن المتوسطات  $a_j$  و  $b_j$  التي تعرف دوال التلييد الثانوية  $h_j$  للمسحوبة من الصف  $\mathcal{H}_{p,m_j}$  الوارد في المقطع 3.11-3 (ما عدا في حالة  $\pi_j = 1$  نستخدم  $a = b = 0$ ). تقدم المبرهنة والنتيجة التاليتان حدًا لحجوم جميع جداول التلييد الثانوية المتوقعة مجتمعًا. ثمة نتيجة ثانية تُحدِّد احتمال أن يكون حجم جميع جداول التلييد الثانوية مجتمعة فوق عطي (يساوي) أو يتجاوز  $4n$  فعليًا).

### مبرهنة 10.11

افترض أننا نخرز  $n$  مفتاحًا في جدول تلييد حجمه  $m = n$  باستخدام دالة تلييد  $h$  مختارة عشوائيًا من صف شامل من دوال تلييد. عندها، يكون لدينا

$$E \left[ \sum_{j=0}^{m-1} \pi_j^2 \right] < 2n.$$

حيث  $n_j$  عدد المفاتيح للتلييد في الشُئْب  $z$ .

**البرهان** سنبدأ بالمطابقة التالية، التي تتحقق في حالة أي عدد صحيح غير سالب  $\alpha$ :

$$\alpha^2 = \alpha + 2 \binom{\alpha}{2} . \quad (6.11)$$

لدينا

$$\begin{aligned} E \left[ \sum_{j=0}^{m-1} n_j^2 \right] &= E \left[ \sum_{j=0}^{m-1} \left( n_j + 2 \binom{n_j}{2} \right) \right] \quad ((\text{من المعادلة (6.11)}) \\ &= E \left[ \sum_{j=0}^{m-1} n_j \right] + 2E \left[ \sum_{j=0}^{m-1} \binom{n_j}{2} \right] \quad (\text{حسب خطية التوقع}) \\ &= E[n] + 2E \left[ \sum_{j=0}^{m-1} \binom{n_j}{2} \right] \quad ((\text{من المعادلة (1.11)}) \\ &= n + 2E \left[ \sum_{j=0}^{m-1} \binom{n_j}{2} \right] . \quad (\text{لأن } n \text{ ليست متحولاً عشوائياً}) \end{aligned}$$

لتقوم المجموع  $\sum_{j=0}^{m-1} \binom{n_j}{2}$ ، نلاحظ أن العدد الكلي لأزواج المفاتيح في جدول التليبد هو الذي يتصادم فقط. وبحسب خواص التليبد الشامل، فإن القيمة للتوقعة هذا المجموع هي على الأكثر

$$\begin{aligned} \binom{n}{2} \frac{1}{m} &= \frac{n(n-1)}{2m} \\ &= \frac{n-1}{2} . \end{aligned}$$

لأن  $m = n$ ، إذن،

$$\begin{aligned} E \left[ \sum_{j=0}^{m-1} n_j^2 \right] &\leq n + 2 \frac{n-1}{2} \\ &= 2n - 1 \\ &< 2n . \end{aligned}$$

■

### نتيجة 11.11

افترض أننا نخزن  $n$  مفاتيحًا في جدول تليبد حجمه  $m = n$  باستخدام دالة تليبد  $h$  مختارة عشوائيًا من صف شامل من دوال التليبد، وأتأمل بحجم كل جدول تليبد ثانوي  $m_j = n_j^2$  حيث  $j = 0, 1, \dots, m-1$ .

إذن يكون الحجم المتوقع من الذاكرة اللازمة لجميع جداول التلييد الثانوية في منهج التلييد الكامل أقل من  $2n$ .

**البرهان** لما كان  $m_j = n_j^2$  حيث  $j = 0, 1, \dots, m-1$  فإن للمبرهنة 10.11 تعطي

$$E \left[ \sum_{j=0}^{m-1} m_j \right] = E \left[ \sum_{j=0}^{m-1} n_j^2 \right]$$

$$< 2n ,$$

■

وبذا يكتمل البرهان.

### نتيجة 12.11

افترض أننا نخزن  $n$  مفتاحاً في جدول تلييد حجمه  $m = n$  باستخدام دالة تلييد  $h$  مختارة عشوائياً من صف شامل من دوال التلييد، وأنها تحمل حجم كل جدول تلييد ثانوي  $m_j = n_j^2$  حيث  $j = 0, 1, \dots, m-1$  عندها يكون احتمال أن تساوي الذاكرة الكلية المستخدمة لجداول التلييد الثانوية قيمة  $4n$  أو تتجاوزها، أقل من  $1/2$ .

**البرهان** نطبق مترابحة ماركوف (ت.30) ثانية،  $E[X]/\varepsilon \leq \Pr\{X \geq \varepsilon\}$ ، ولكن هذه المرة على المترابحة (7.11)، حيث  $X = \sum_{j=0}^{m-1} m_j$  و  $\varepsilon = 4n$ :

$$\begin{aligned} \Pr \left\{ \sum_{j=0}^{m-1} m_j \geq 4n \right\} &\leq \frac{E \left[ \sum_{j=0}^{m-1} m_j \right]}{4n} \\ &< \frac{2n}{4n} \\ &= \frac{1}{2} . \end{aligned}$$

■

نلاحظ من النتيجة 12.11، أننا إذا فحصنا القليل من دوال التلييد المختارة عشوائياً من جماعة شاملة، فسنجد سريعاً دالة تلييد تستخدم حجماً معقولاً من الذاكرة.

تعاريف

### ★ 1-5.11

افترض أننا ندرج  $n$  مفتاحاً في جدول تلييد حجمه  $m$  باستخدام العنونة المفتوحة وتلييد منتظم. ولكن  $p(n, m) \leq e^{-n(n-1)/2m}$ . (لمنح: انظر للمعادلة (12.3)). ناقش أنه عندما تتجاوز  $n$  القيمة  $\sqrt{m}$ ، يتأهي احتمال تجنب التصادمات إلى الصفر بسرعة.



## مسائل

## 1-11 حد أطول سِر التلييد

افترض أننا نستخدم جدول تلييد ذا عتونة مفتوحة حجمه  $m$  لتخزين  $n \leq m/2$  بنداً.

أ. بفرض أن التلييد منتظم، بيّن، في حالة  $i = 1, 2, \dots, n$ ، أن احتمال أن تحتاج عملية الإدراج ذات الترتيب  $i$  إلى أكثر من  $k$  سِراً بالضبط هو  $2^{-k}$  على الأكثر.

ب. بيّن في حالة  $i = 1, 2, \dots, n$ ، أنَّ احتمال أن تحتاج عملية الإدراج ذات الترتيب  $i$  إلى أكثر من  $2 \lg n$  سِراً هو  $O(1/n^2)$ .

لنشير بـ  $X_i$  إلى المتحول العشوائي الذي يرمز إلى عدد السُور [جميع سُر] اللازمة لعملية الإدراج ذات الترتيب  $i$ . وقد وجدت في الجزء (ب) أن  $\Pr\{X_i > 2 \lg n\} = O(1/n^2)$ . افترض أن المتحول العشوائي  $X = \max_{1 \leq i \leq n} X_i$  يرمز للمعدن الأعظمي للسور اللازمة لـ  $n$  عملية إدراج.

ت. بيّن أن  $\Pr\{X > 2 \lg n\} = O(1/n)$ .

ث. بيّن أن الطول المتوقع  $E[X]$  لأطول متالية سِر هو  $O(\lg n)$ .

## 2-11 حد حجم شُقب عند استخدام السلسلة

افترض أن لدينا جدول تلييد يحتوي  $n$  شُقباً، وأنه جرى تميز التصادمات بالسلسلة، وافترض أنه جرى إدراج  $n$  مفتاحاً في الجدول. لكل مفتاح احتمال مناسب في أن يُلد في أي شُقب. وليكن  $M$  عدد المفاتيح الأعظم في أي شُقب بعد أن تُدرج كل المفاتيح. أثبت أن القيمة العظمى لـ  $E[M]$ ، توفُّع  $M$ ، المتوقعة هي  $O(\lg n / \lg \lg n)$ .

أ. ناقش أن يكون الاحتمال  $Q_k$ ، لتليد  $k$  مفتاحاً تماماً في شُقب محدّد يعطى بالعلاقة

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k}.$$

ب. ليكن  $P_k$  احتمال أن تكون  $M = k$ ، أي، احتمال أن يكون الشُقب الذي يحتوي غالبية المفاتيح يحتوي  $k$  مفتاحاً. بيّن أن  $P_k \leq nQ_k$ .

ت. استخدم تقريب Stirling، المعادلة (3.18)، لبيان أن  $Q_k < e^k/k^k$ .

ث. بيّن أنه يوجد ثابت  $c > 1$  بحيث  $Q_{k_0} < 1/n^3$  حيث  $k_0 = c \lg n / \lg \lg n$ . استنتج أن  $P_k < 1/n^2$  لكل  $k \geq k_0 = c \lg n / \lg \lg n$ .

ج. ناقش أن

$$E[M] \leq \Pr\left\{M > \frac{c \lg n}{\lg \lg n}\right\} \cdot n + \Pr\left\{M \leq \frac{c \lg n}{\lg \lg n}\right\} \cdot \frac{c \lg n}{\lg \lg n}.$$

واستنتج أن  $E[M] = O(\lg n / \lg \lg n)$ .

### 3-11 السير التريمي

افترض أنه قد طُلب إلينا البحث عن مفتاح  $k$  في جدول تليد موافق  $1, 2, \dots, m$ ، وافترض أن لدينا دالة تليد  $h$  تطابق فضاء المفتاح إلى المجموعة  $\{0, 1, \dots, m-1\}$ . يكون فتح البحث كالتالي:

1. احسب القيمة  $h(k)$ ، وضع  $j = 0$ .
  2. اسر للوقع  $i$  للمفتاح المرغوب  $k$ . أنه البحث إذا وجدته، أو إذا كان هذا الموقع فارغاً.
  3. ضغ  $i = i + 1$ . إذا كان  $i$  يساوي الآن  $m$ ، يكون الجدول ممتلئاً، لذلك أنه البحث، وإلا، ضغ  $i = (i + j) \bmod m$ ، ثم عد إلى الخطوة 2.
- افترض أن  $m$  من قوى العدد 2.

أ. بَيِّن أن هذا المخطط ينتج instance عن مخطط "السير التريمي" العام بإعطاء الثوابت المناسبة  $c_1$  و  $c_2$  للمعادلة (5.11).

ب. أثبت أن هذه الخوارزمية في أسوأ الحالات تفحص كل موقع في الجدول.

### 4-11 التليد والاستيقان Authentication

ليكن  $\mathcal{H}$  صفًا من دوال التليد تُقابل فيه كل دالة تليد  $h \in \mathcal{H}$  المجموعة الشاملة من المفاتيح  $U$  إلى  $\{0, 1, \dots, m-1\}$ . نقول إن  $\mathcal{H}$  هو شامل من الطول  $k$ -universal إذا تحقق أنه، لكل متتالية ثابتة مكونة من  $k$  مفتاحاً متمايزاً  $\langle x^{(1)}, x^{(2)}, \dots, x^{(k)} \rangle$  وأي  $h$  مختار عشوائياً من متتالية  $\mathcal{H}$ ، فإن المتتالية  $\langle h(x^{(1)}), h(x^{(2)}), \dots, h(x^{(k)}) \rangle$  منسوبة الاحتمال في أن تكون أيًا من  $m^k$  متتالية من الطول  $k$  حيث تُسحب العناصر من  $\{0, 1, \dots, m-1\}$ .

- أ. بَيِّن أنه إذا كانت جماعة دوال التليد  $\mathcal{H}$  شاملة من الطول 2-universal، فإن  $\mathcal{H}$  شاملة.
- ب. افترض أن المجموعة الشاملة  $U$  هي مجموعة مكونة من قيم ذات  $n$  مركبة، مسحوبة من  $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$  حيث  $p$  أولي. خذ عنصرًا  $x \in U$   $x = \langle x_0, x_1, \dots, x_{n-1} \rangle$  عرّف لأي  $n$  مركبة  $a = \langle a_0, a_1, \dots, a_{n-1} \rangle \in U$  دالة التليد  $h_a$  حيث:

$$h_a(x) = \left( \sum_{j=0}^{n-1} a_j x_j \right) \bmod p .$$

ليكن الصف  $\mathcal{H} = \{h_a\}$  . يَبَيَّنُ أَنَّ  $\mathcal{H}$  شاملٌ، ولكن ليس شاملاً من الطول 2 -universal .  
(تلميح: أوجد مفتاحاً تُنتِجُ له جميع دوال التلييد في  $\mathcal{H}$  القيمة نفسها.)

ت. افترض أننا عدلنا  $\mathcal{H}$  قليلاً من الجزء (ب): بحيث يكون لأي  $u \in U$  ولايٍ عنصر  $b \in \mathbb{Z}_p$  نعرّف

$$h'_{ab}(x) = \left( \sum_{j=0}^{n-1} a_j x_j + b \right) \bmod p$$

و  $\mathcal{H}' = \{h'_{ab}\}$  ناقش كون  $\mathcal{H}'$  شاملاً من الطول 2 . (تلميح: افترض عنصرين ثابتين  $x, y \in U$  بحيث  $x_i \neq y_i$  لقيمة ما  $i$  . ماذا يحصل لكل من  $h'_{ab}(x)$  و  $h'_{ab}(y)$  حين تتغير قيم  $a_i$  و  $b$  ضمن  $\mathbb{Z}_p$ ؟)

ث. افترض أن Alice و Bob اتفقا سراً على دالة تلييد  $h$  من جماعة من دوال التلييد  $\mathcal{H}$  شاملة ذات الطول 2 . كل دالة  $h \in \mathcal{H}$  تقابل بين فضاء المفاتيح  $U$  و  $\mathbb{Z}_p$  ، حيث  $p$  أٌقْبِي . فبما بعد، ترسل Alice رسالة  $m$  إلى Bob بواسطة الإنترنت، حيث  $m \in U$  . وستُيَقَّنُ هذه الرسالة لـ Bob أيضاً بإرسال لمعية استيقان  $\epsilon = h(m)$  ، ويفحص Bob كون الزوج  $(m, \epsilon)$  الذي استقبله يحقق  $\epsilon = h(m)$  . افترض أن خصمنا سيعرض مسار  $(m, \epsilon)$  ويحاول أن يخدع Bob بالاستعاضة عن هذا الزوج بزوج آخر  $(m', \epsilon')$  . ناقش أن يكون احتمال أن ينجح الخصم في خداع Bob بقبول الزوج  $(m', \epsilon')$  هو على الأكثر  $1/p$  ، بقطع النظر عن قدرة الحساب التي يمتلكها الخصم، وحتى لو كان الخصم يعرف عائلة دوال التلييد المستخدمة  $\mathcal{H}$  .

## ملاحظات الفصل

Knuth [211] و Gonnet [145] مرجعان ممتازان عن تحليل خوارزميات التلييد . يُرجعُ Knuth اختراع جداول التلييد وطريقة السلسلة في تمييز التصاديات إلى H. P. Luhn (1953) . وفي الوقت نفسه تقريباً أوجد G. M. Amdahl فكرة العنونة المفتوحة .

قدّم Carter و Wegman مفهوم الصفوف الشاملة لدوال التلييد في عام 1979 [59] .

طُوِّرَ كُلُّ من Fredman و Komlós و Szemerédi [112] منهج التلييد الكامل للمجموعات الساكنة الذي عرض في المقطع 5.11 . ووشع Dietzfelbinger وآخرون [87] طريقتهم للمجموعات الديناميكية، ومعالجة عمليات الإدراج والحذف في زمن مُتَوَقَّع مُتَحَدٍ  $O(1)$  .

## 12 أشجار البحث الثنائية

تدعم بنية معطيات أشجار البحث الكثير من العمليات على المجموعة الديناميكية، ويشمل ذلك عملية البحث (SEARCH)، وإيجاد القيمة الصغرى (MINIMUM)، وإيجاد القيمة العظمى (MAXIMUM)، وإيجاد السابق (PREDECESSOR)، وإيجاد اللاحق (SUCCESSOR)، وعملية الإدراج (INSERT) والحذف (DELETE). وبذلك، يمكننا استخدام شجرة البحث كمعجم وكرنل ذو أولوية (priority queue) على حد سواء.

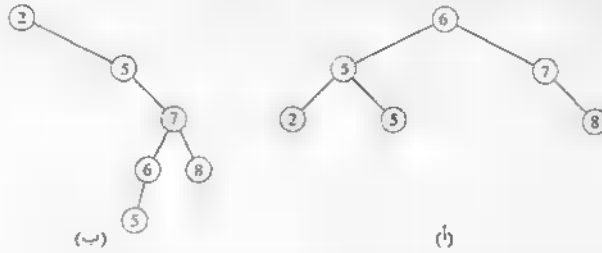
تستغرق العمليات الأساسية على شجرة بحث ثنائية زمناً يتناسب مع ارتفاع الشجرة. وفي حالة شجرة ثنائية كاملة من  $n$  عقدة، يستغرق زمن تنفيذ مثل هذه العمليات  $\Theta(\lg n)$  في أسوأ الحالات. ولكن إذا كانت الشجرة سلسلة خطية من  $n$  عقدة، فإن العمليات نفسها تستغرق زمناً  $\Theta(n)$  في أسوأ الحالات. وسنرى في المقطع 4.12 أن الارتفاع المتوقع لشجرة بحث ثنائية مبنية عشوائياً هو  $O(\lg n)$ ، وبذلك تستغرق العمليات على المجموعة الديناميكية الأساسية على شجرة كهذه زمناً وسطيّاً  $\Theta(\lg n)$ .

ومن الناحية العملية، لا يمكننا دوماً أن نضمن أن تكون أشجار البحث الثنائية قد بُنيت عشوائياً، إلا أنه يمكننا إيجاد متغيرات (نسخ معدلة) من أشجار البحث الثنائية نضمن أن يكون أدائها على العمليات الأساسية في أسوأ الحالات، جيّداً. يعرض الفصل 13 مثلاً عن مثل هذه المتغيرات، وهو الأشجار الحمراء-السوداء (red-black trees)، التي ارتفاعها  $O(\lg n)$ . ويعرض الفصل 18 الأشجار للعممة (B-trees)، التي تُعتبر جيدة بصفة خاصة للحفاظ على قواعد المعطيات على قرص خزن ناووي.

بعد عرض الخصائص الأساسية لأشجار البحث الثنائية، تبين للمقاطع التالية كيف يمكن أن نجوّب شجرة بحث ثنائية لطباعة قيمها بترتيب مفروز، وكيف نبحث عن قيمة في شجرة بحث ثنائية، وكيف نوجد العنصر الأصغر أو الأكبر، وكيف نوجد للاحق عنصر ما وسابقه، وكيف ندرج في شجرة بحث ثنائية أو نحذف منها. يُظهر الملحق ب- الخصائص الرياضية الأساسية للأشجار.

### 1.12 ما هي شجرة البحث الثنائية؟

تنظم شجرة البحث الثنائية، كما يوحي اسمها، على شكل شجرة ثنائية، كما هو مبين في الشكل 1.12. يمكننا تمثيل هذه الشجرة ببنية معطيات مترابطة تُولف كل عقدة فيها غرضاً. وتحتوي كل عقدة، إضافة إلى



**الشكل 1.12** أشجار بحث ثنائية. إن للمفاتيح في الشجرة الفرعية اليسرى لأي عقدة  $x$  تساوي على الأكثر  $x.key$ ، والمفاتيح في الشجرة الفرعية اليمنى من  $x$  تساوي  $x.key$  على الأقل. يمكن أن تمثل أشجار بحث ثنائية مختلفة مجموعة القيم نفسها. يتناسب زمن التنفيذ، في أسوأ الحالات، لأغلب عمليات البحث في الشجرة، مع ارتفاع الشجرة. (أ) شجرة بحث ثنائية من 6 عقد ارتفاعها 2. (ب) شجرة بحث ثنائية أقل فعالية ارتفاعها 4 وتحتوي المفاتيح نفسها.

المفتاح  $key$  والمعلومات التابعة  $satellite data$ ، على الواصفات يسار  $left$ ، ويمين  $right$  و  $p$  التي تشير إلى العقد المقابلة للابن الأيسر وللأبن والأيمن وللأب، على الترتيب. عندما يغيب الابن أو الأب، يحتوي الوصف الموافق القيمة NIL. إن عقدة الجذر root node هي العقدة الوحيدة في الشجرة التي تمتلك حقل أب يساوي NIL.

تُخزن للمفاتيح في شجرة بحث ثنائية دونما بحيث تحقق خاصية شجرة البحث الثنائية

**binary-search-tree property:**

لتكن  $x$  عقدة في شجرة بحث ثنائية. فإذا كانت  $y$  عقدة في الشجرة الفرعية اليسرى لـ  $x$  فإن

$$y.key \leq x.key \quad \text{وإذا كانت } y \text{ عقدة في الشجرة الفرعية اليمنى لـ } x, \text{ فإن } y.key \geq x.key$$

وهكذا فإن مفتاح الجذر، في الشكل 1.12(أ)، يساوي 6، وللمفاتيح 2 و 5 و 5 في شجرتي الفرعية اليسرى ليست أكبر من 6. والمفاتيح 7 و 8 في الشجرة الفرعية اليمنى ليست أصغر من 6. وهذه الخاصية نفسها محققة لكل عقدة في الشجرة. على سبيل المثال، إن للمفتاح 5 في الابن الأيسر للجذر ليس أصغر من المفتاح 2 في الشجرة الفرعية اليسرى لتلك العقدة وليس أكبر من المفتاح 5 في الشجرة الفرعية اليمنى.

نسمح لنا بخاصية شجرة البحث الثنائية بطباعة كل للمفاتيح في شجرة البحث الثنائية بترتيب مغزول باستخدام خوارزمية عودية بسيطة، تسمى *تجوال في الشجرة وفق الترتيب البيني inorder tree walk*. تمثت هذه الخوارزمية هكذا لكونها تطبع مفتاح جذر شجرة فرعية بين طباعة القيم الموجودة في شجرتها

الفرعية اليسرى وطباعة تلك الموجودة في شجرة الفرعية اليمنى. (بالطريقة نفسها، قطع خوارزمية *تجوال في الشجرة وفق الترتيب السبقي preorder tree walk* الجذر قبل القيم في أي من الشجرتين الفرعيتين، وتقطع خوارزمية *تجوال في الشجرة وفق الترتيب اللحقي postorder tree walk* الجذر بعد القيم في أشجارها الفرعية.) ولاستعمال الإجراء التالي لطباعة جميع العناصر في شجرة بحث ثنائية  $T$ ، فإننا نستدعي  $\text{INORDER-TREE-WALK}(T, \text{root})$ .

**INORDER-TREE-WALK( $x$ )**

```

1  if  $x \neq \text{NIL}$ 
2      INORDER-TREE-WALK( $x.\text{left}$ )
3      print  $x.\text{key}$ 
4      INORDER-TREE-WALK( $x.\text{right}$ )

```

على سبيل المثال، نقطع خوارزمية *تجوال في شجرة وفق الترتيب الداخلي* المفاتيخ في كل من شجري البحث الثنائية من الشكل 1.12 بالترتيب 2، 5، 5، 6، 7، 8. وتُستدعى صحة الخوارزمية، مباشرة، من خاصية شجرة-البحث-الثنائية.

تستغرق الخوارزمية زمناً  $\Theta(n)$  لتجوب شجرة بحث ثنائية من  $n$  عقدة، لأن الإجراء، بعد الاستدعاء الأولي، يستدعي نفسه على نحوٍ عَوْدِي مرتين بالضبط لكل عقدة في الشجرة؛ مرةً في حالة ابنها الأيسر ومرةً في حالة ابنها الأيمن. وتُقدم للموهنة التالية برهاناً منهجياً على أن الخوارزمية تستغرق زمناً خطياً لإنجاز *تجوال في شجرة وفق الترتيب البيني*.

### مبرهنة 1.12

إذا كان  $x$  جذراً لشجرة فرعية من  $n$  عقدة، فإن استدعاء  $\text{INORDER-TREE-WALK}(x)$  يستغرق زمناً  $\Theta(n)$ .

**البرهان** يُنشر بـ  $T(n)$  إلى الزمن الذي تستغرقه  $\text{INORDER-TREE-WALK}$  لدى استدعائها عند جذر شجرة فرعية من  $n$  عقدة. لذا كان  $\text{INORDER-TREE-WALK}$  يزور جميع العقد  $n$  للشجرة الفرعية، كان لدينا  $T(n) = \Omega(n)$ . يبقى علينا إثبات أن  $T(n) = O(n)$ .

ولمّا كان  $\text{INORDER-TREE-WALK}$  يستغرق مدّة ثابتة وصغيرة في شجرة فرعية فارغة (عند اختبار  $x \neq \text{NIL}$ )، فإن  $T(0) = c$  حيث  $c > 0$  ثابت ما.

لنفترض، في حالة  $n > 0$ ، أنه جرى استدعاء الإجراء  $\text{INORDER-TREE-WALK}$  عند العقدة  $x$  التي لشجرتها الفرعية اليسرى  $k$  عقدة ولشجرتها الفرعية اليمنى  $n - k - 1$  عقدة. إن الزمن اللازم لإنجاز  $\text{INORDER-TREE-WALK}(x)$  يعود بـ  $T(k) + T(n - k - 1) + d$ ، حيث  $d$  ثابت ما

موجب تمامًا ( $d > 0$ )، يعكس هذا أعلى الزمن لتنفيذ متري الإجراء  $\text{INORDER-TREE-WALK}(x)$  باستثناء الزمن المستغرق في الاستدعاءات العودية.

نستخدم طريقة التعميض لنثبت أن  $T(n) = O(n)$  وذلك ببرهان أن  $T(n) \leq (c + d)n + c$ . من أجل  $n = 0$ ، لدينا  $c = T(0) = c + d \cdot 0 + c$ . ومن أجل  $n > 0$ ، لدينا:

$$\begin{aligned} T(n) &\leq T(k) + T(n - k - 1) + d \\ &= ((c + d)k + c) + ((c + d)(n - k - 1) + c) + d \\ &= (c + d)n + c - (c + d) + c + d \\ &= (c + d)n + c, \end{aligned}$$

وهذا هو المطلوب. ■

### تمارين

#### 1-1.12

ارسم أشجار بحث ثنائية بارتفاع 2 و 3 و 4 و 5 و 6 مجموعة المفاتيح {1, 4, 5, 10, 16, 17, 21}.

#### 2-1.12

ما الفرق بين خاصية شجرة-البحث-الثنائية وخاصية الكومة وفق الأصغر min-heap (انظر الصفحة 154)؟ هل يمكن استخدام خاصية الكومة وفق الأصغر لطباعة مفاتيح شجرة من  $n$  عقدة بترتيب مفروز في زمن  $O(n)$ ؟ اشرح الكيفية، أو اشرح لماذا لا يمكن ذلك.

#### 3-1.12

أعط خوارزمية غير عودية تُنفَّذ نحواً في شجرة وفق الترتيب البيني. (تلميح: في حل سهل يُستعمل مكدهس كبنية معطيات رديفة. وفي حل أكثر تعقيداً، لكنه أنيق، لا يُستعمل مكدهس، وإنما يفترض أن بإمكاننا اختبار حالة تساوي مؤشرين.)

#### 4-1.12

أعط خوارزمية عودية تُنفَّذ نحواً في شجرة بالترتيب السبتي وبالترتيب اللحقفي في زمن  $\Theta(n)$  على شجرة من  $n$  عقدة.

#### 5-1.12

ناقش مايلي: لماذا كان فرز  $n$  عنصراً يستغرق في أسوأ الحالات زمناً  $\Omega(n \lg n)$  وفق نموذج المقارنة comparison model، فإن أي خوارزمية تعتمد على المقارنة لبناء شجرة بحث ثنائية من لائحة اعتباطية من  $n$  عنصراً تستغرق في أسوأ الحالات زمناً  $\Omega(n \lg n)$ .

## 2.12 استعمال شجرة بحث ثنائية

كثيراً ما نحتاج إلى البحث عن مفتاح مخزن في شجرة بحث ثنائية. فإضافة إلى عملية البحث SEARCH، يمكن أن تدعم أشجار البحث الثنائية استعلامات مثل الأصغر MINIMUM، الأكبر MAXIMUM، اللاحق SUCCESSOR، السابق PREDECESSOR. وسندرس، في هذا المقطع، هذه العمليات ونبين كيف يمكن تنفيذ كل منها في زمن  $O(h)$ ، على شجرة بحث ثنائية ما ارتفاعها  $h$ .

### البحث

نستخدم الإجراء التالي للبحث عن العقدة التي تحتوي مفتاحاً محدداً في شجرة بحث ثنائية. فإذا كان لدينا مؤشر إلى جذر الشجرة ومفتاح  $k$ ، فإن إجراء البحث في الشجرة TREE-SEARCH يعيد مؤشراً إلى عقدة مفتاحها  $k$  إن وجدت، وإلا فإنه يعيد NIL.

```

TREE-SEARCH( $x, k$ )
1  if  $x == \text{NIL}$  or  $k == x.\text{key}$ 
2      return  $x$ 
3  if  $k < x.\text{key}$ 
4      return TREE-SEARCH( $x.\text{left}, k$ )
5  else return TREE-SEARCH( $x.\text{right}, k$ )

```

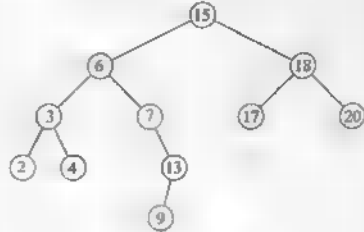
تبدأ الإجراءية بحثها من الجذر وتتبع مساراً بسيطاً نازلاً باتجاه أسفل الشجرة، كما هو مبين في الشكل 2.12. ثم تقارن، لكل عقدة  $x$  تصادفها، المفتاح  $k$  بالمفتاح  $x.\text{key}$ . فإذا تساوى المفتاحان انتهى البحث. وإذا كان  $k$  أصغر من  $x.\text{key}$ ، استمر البحث في الشجرة الفرعية اليسرى لـ  $x$ ، لأن خاصية شجرة-البحث-الثنائية تقتضي عدم إمكان تخزين  $k$  في الشجرة الفرعية اليمنى. وبالطريقة نفسها، إذا كان  $k$  أكبر من  $x.\text{key}$ ، استمر البحث في الشجرة الفرعية اليمنى. تُشكل العقد التي تُصادف خلال العودة مساراً بسيطاً نازلاً من جذر الشجرة، ومن ثم فإن زمن تنفيذ TREE-SEARCH هو  $O(h)$ ، حيث  $h$  ارتفاع الشجرة. يمكننا كتابة الإجراء نفسه على نحو تكراري = "نشر unrolling" العودية إلى حلقة while. وهذه النسخة أكثر فعالية على معظم الحواسيب.

```

ITERATIVE-TREE-SEARCH( $x, k$ )
1  while  $x \neq \text{NIL}$  and  $k \neq x.\text{key}$ 
2      if  $k < x.\text{key}$ 
3           $x = x.\text{left}$ 
4      else  $x = x.\text{right}$ 
5  return  $x$ 

```





**الشكل 2.12** استعلامات على شجرة بحث ثنائية. للبحث عن المفتاح 13 في الشجرة فإننا نتبع المسار  $15 \leftarrow 6 \leftarrow 7 \leftarrow 13$  بدءاً من الجذر. إن المفتاح الأصفر في الشجرة هو 2، ونجدّه بتعقّب المؤشرات اليسرى *left pointers* من الجذر. والمفتاح الأكبر هو 20، ونجدّه بتعقّب المؤشرات اليمنى *right pointers* من الجذر. إن لاحق العقدة ذات المفتاح 15 هي العقدة ذات المفتاح 17، لكونه المفتاح الأصفر في الشجرة الفرعية اليمنى لـ 15. لا تحتلّ العقدة ذات المفتاح 13 شجرة فرعية يميني، ومن ثمّ فإنّ لاحقها هو سلفها الأدنى الذي ابنه الأيسر هو أيضاً سلف. وفي هذه الحالة، تكون العقدة ذات المفتاح 15 هي لاحقها.

### الأصفر والأكبر

يمكننا دومًا العثور على عنصر في شجرة بحث ثنائية مفتاحها أصفر بتتبع مؤشرات الأبناء اليسرى من الجذر إلى أن تُصادف القيمة *NIL*، كما هو مبين في الشكل 2.12. يعيد الإجراء التالي مؤشرًا إلى العنصر الأصفر في شجرة فرعية جذرها عند عقدة معطاة *x*، والذي نفترضه لا يساوي *NIL*:

```

TREE-MINIMUM(x)
1  while x.left ≠ NIL
2     x = x.left
3  return x
  
```

تضمن خاصية شجرة-البحث-الثنائية صحة الإجراء *TREE-MINIMUM*. فإذا لم يكن للعقدة *x* شجرة فرعية يسرى، فإن للمفتاح الأصفر في الشجرة الفرعية التي جذرها *x* هو *x.key*، لأن كل مفتاح في الشجرة الفرعية اليمنى لـ *x* هو على الأقل بأكبر *x.key*. أما إذا كان للعقدة *x* شجرة فرعية يسرى، فإن المفتاح الأصفر في الشجرة الفرعية التي جذرها *x* يقع في الشجرة الفرعية التي جذرها عند *x.left*، لأنه لا يوجد مفتاح في الشجرة الفرعية اليمنى أصغر من *x.key* وكلّ مفتاح في الشجرة الفرعية اليسرى ليس أكبر من *x.key*.

إن شبه الرمز لـ *TREE-MAXIMUM* مشابه لـ *TREE-MINIMUM*.

```

TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2     x = x.right
3  return x
  
```

يُنفَّذُ كلا الإجراءين في زمن  $O(h)$  لشجرة ارتفاعها  $h$ ، لأن متتالية العقد العارضة، كما في إجراء TREE-SEARCH، تشكل مسارًا بسيطًا نازلًا من جذر الشجرة.

### اللاحق والسابق

إذا كانت لدينا عقدة في شجرة بحث ثنائية، فقد نحتاج أحيانًا إلى إيجاد لاحقها في الترتيب المفروز المُحدد بواسطة تحوال في شجرة وفق الترتيب البيني. فإذا كانت جميع المفاتيح متميزة، فإن لاحق العقدة  $x$  هي العقدة ذات أصغر مفتاح أكبر من  $x.key$ . تسمح لنا بنية شجرة البحث الثنائية بتحديد لاحق عقدة حتى بدون مقارنة المفاتيح. ويحدد الإجراء التالي لاحق العقدة  $x$  في شجرة بحث ثنائية إن كان موجودًا، ويعيد NIL إن كان مفتاح  $x$  هو الأكبر في الشجرة.

TREE-SUCCESSOR( $x$ )

```

1  if  $x.right \neq NIL$ 
2      return TREE-MINIMUM( $x.right$ )
3   $y = x.p$ 
4  while  $y \neq NIL$  and  $x == y.right$ 
5       $x = y$ 
6       $y = y.p$ 
7  return  $y$ 
```

نحزك رماز الإجراء TREE-SUCCESSOR إلى حالتين. إذا لم تكن الشجرة الفرعية اليمنى للعقدة  $x$  فارغة، فإن لاحق  $x$  هي غاما أقصى عقدة يسرى leftmost في الشجرة الفرعية اليمنى، والتي نجدها في السطر 2 باستدعاء TREE-MINIMUM( $x.right$ ). على سبيل للنال، إن لاحق العقدة ذات المفتاح 15 في الشكل 2.12 هي العقدة ذات المفتاح 17.

من ناحية أخرى، وكما يُطلب إليك في التمرين 6-2.12 بيانه، إذا كانت الشجرة الفرعية اليمنى للعقدة  $x$  فارغة وكان لـ  $x$  لاحق  $y$ ، فإن  $y$  هو السلف الأدنى لـ  $x$  الذي ابنه الأيسر هو أيضًا سلف لـ  $x$ . إن لاحق العقدة ذات المفتاح 13، في الشكل 2.12، هي العقدة ذات المفتاح 15. ولإيجاد  $y$ ، ما علينا إلا أن نحدد الشجرة ابتداءً من  $x$  حتى نلاقي عقدة تكون هي الابن الأيسر لأبيها؛ تعالج الأسطر 3-7 في الإجراء TREE-SUCCESSOR هذه الحالة.

إن زمن تنفيذ TREE-SUCCESSOR على شجرة ارتفاعها  $h$  هو  $O(h)$ ، وذلك لأننا إما أن نتبع مسارًا بسيطًا باتجاه أعلى الشجرة، وإما أن نتبع مسارًا بسيطًا باتجاه أسفل الشجرة. إن الإجراء TREE-PREDECESSOR، المناظر لـ TREE-SUCCESSOR، يُنفَّذُ أيضًا في زمن  $O(h)$ . وحتى إن لم تكن المفاتيح متميزة، فإننا نعرّف لاحق أي عقدة  $x$  وسابقتها بالعقدة المُعاداة من جراء الاستدعاءين TREE-SUCCESSOR( $x$ ) و TREE-PREDECESSOR( $x$ ) على الترتيب.

بإيجاز، نكون قد برهنا المبرهنة التالية.

### مبرهنة 2.12

يمكننا تنجيز العمليات على مجموعة ديناميكية: البحث، والأصفر، والأكبر، واللاحق والسابق بحيث تُنفَّذ كل منها في زمن  $O(h)$  على شجرة بحث ثنائية ارتفاعها  $h$ .

### تمارين

#### 1-2.12

بافتراض أن لدينا أعداداً بين 1 و 1000 في شجرة بحث ثنائية، ونريد البحث عن العدد 363. أيٌّ من المتواليات الآتية لا يمكن أن تكون متتالية من العقد للمدروسة؟

أ. 2, 252, 401, 398, 330, 344, 397, 363

ب. 924, 220, 911, 244, 898, 258, 362, 363

ت. 925, 202, 911, 240, 912, 245, 363

ث. 2, 399, 387, 219, 266, 382, 381, 278, 363

ج. 935, 278, 347, 621, 299, 392, 358, 363

#### 2-2.12

اكتب نسختين غُودية لـ TREE-MINIMUM و TREE-MAXIMUM.

#### 3-2.12

اكتب إجراء البحث عن السابق TREE-PREDECESSOR.

#### 4-2.12

يعتقد البرفسور بُنيان Bunyan أنه قد اكتشف خاصية مدهشة لأشجار البحث الثنائية. افترض أن البحث عن مفتاح  $k$  في شجرة بحث ثنائية قد انتهى عند ورقة leaf. تأمل ثلاث مجموعات:  $A$ ، المفاتيح الواقعة إلى يسار مسار البحث؛ و  $B$ ، المفاتيح الواقعة على مسار البحث؛ و  $C$ ، المفاتيح الواقعة إلى يمين مسار البحث. يدعي البرفسور أن أي ثلاثة مفاتيح  $a \in A$  و  $b \in B$  و  $c \in C$  يجب أن تُعقّق  $a \leq b \leq c$ . أعطِ أصغر مثال معاكس يمكن أن يدحض ادعاء البرفسور بُنيان.

#### 5-2.12

بيّن أنه إذا كان لعقدة في شجرة بحث ثنائية ابنان، فليس للاحقها ابنٌ أيسر وليس لسابقها ابنٌ أكبر.

#### 6-2.12

ادرس شجرة بحث ثنائية  $T$  مفاتيحها متميزة. بيّن أنه إذا كانت الشجرة الفرعية اليمنى للعقدة  $x$  في الشجرة

$T$  فارغة وكان  $y$  لاحق لـ  $x$ ، فإن  $y$  هي السلف الأدنى lowest ancestor لـ  $x$  الذي ابنه الأيسر هو أبنا سلف لـ  $x$ . (تذكر أن كل عقدة هي سلف نفسها).

### 7-2.12

ثمة طريقة أخرى لتتجز بحوال بيبي في شجرة بحث ثنائية مؤلفة من  $n$  عقدة، وذلك بإيجاد العنصر الأصغر في الشجرة باستدعاء TREE-MINIMUM، ومن ثم إجراء  $1 -$  استدعاء لـ TREE-SUCCESSOR. برهن أن هذه الخوارزمية تنفذ في زمن  $\Theta(n)$ .

### 8-2.12

برهن أنه إذا كانت العقدة التي تبدأ منها في شجرة بحث ثنائية ارتفاعها  $h$ ، فإن  $k$  استدعاء متتاليًا لـ TREE-SUCCESSOR يستغرق زمنًا  $O(k + h)$ .

### 9-2.12

لتكن  $T$  شجرة بحث ثنائية، مفاتيحها متمايزة، ولتكن  $x$  عقدة ورقة وليكن  $y$  أباه. يَبَيِّن أن  $y.key$  هو إما المفتاح الأصغر في  $T$  الأكبر من  $x.key$ ، وإما هو المفتاح الأكبر في  $T$  الأصغر من  $x.key$ .

## 3.12 الإدراج والحذف

تُسَبِّب عمليتا الإدراج والحذف تغيير المجموعة الديناميكية الشَّخْطَة بشجرة بحث ثنائية. ولا بد من تعديل بنية المعطيات كي يظهر هذا التغيير، ولكن بطريقة تسمح بالمحافظة على خاصية شجرة-البحث-الثنائية. وسنرى لاحقًا أن تعديل الشجرة لإدراج عنصر جديد هي عملية مباشرة نسبيًا، ولكن معالجة الحذف عملية أكثر تعقيدًا نوعًا ما.

### الإدراج

نستخدم إجراء TREE-INSERT لإدراج قيمة جديدة  $v$  في شجرة بحث ثنائية  $T$ . يأخذ الإجراء العقدة  $z$  التي لها  $z.key = v$  و  $z.left = NIL$  و  $z.right = NIL$ . يُعَدَّل الإجراء الشجرة  $T$  وبعض واصفات  $z$  بحيث يُدرج  $z$  في مكان مناسب في الشجرة.

TREE-INSERT( $T, z$ )

- 1  $y = NIL$
- 2  $x = T.root$
- 3 **while**  $x \neq NIL$
- 4      $y = x$
- 5     **if**  $z.key < x.key$
- 6          $x = x.left$

```

7  else x = x.right
8  z.p = y
9  if y == NIL
10     T.root = z // Tree T was empty
11  elseif z.key < y.key
12     y.left = z
13  else y.right = z

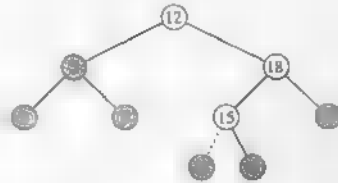
```

يبين الشكل 3.12 آلية عمل الإجراء TREE-INSERT. يبدأ الإجراء TREE-INSERT، كما هو الحال في الإجراءيتين TREESEARCH و ITERATIVE-TREE-SEARCH، من جذر الشجرة ويرسم المؤشر  $x$  مساراً بسيطاً نازلاً باحثاً عن NIL ليستعوض عنه بعنصر الدخول  $z$ . يحافظ الإجراء على مؤشر الأثر  $y$  trailing pointer كإشارة لـ  $x$ . بعد الاستبداء، تسبب حلقة while في الأسطر 3-7 تحريك هذين المؤشرين باتجاه أسفل الشجرة، ويتجهان مساراً أو يمينا اعتماداً على نتيجة مقارنة  $z.key$  بـ  $x.key$ ، حتى يصبح  $x$  مساوياً NIL. يشغل NIL هذا المكان الذي نرغب وضع العنصر  $z$  فيه. ونحتاج إلى مؤشر الأثر  $y$ ، لأننا عندما نحدد الـ NIL التي ينتمي إليها  $z$ ، يكون البحث قد تقدم خطوة واحدة إلى ما بعد العقدة التي نحتاج إلى تغيير. نعي الأسطر 8-13 للمؤشرات التي تسبب إدراج  $z$ .

وعلى نحو مماثل للعمليات الأولية الأخرى على أشجار البحث، يُنفَّذ الإجراء TREE-INSERT في زمن  $O(h)$  على شجرة ارتفاعها  $h$ .

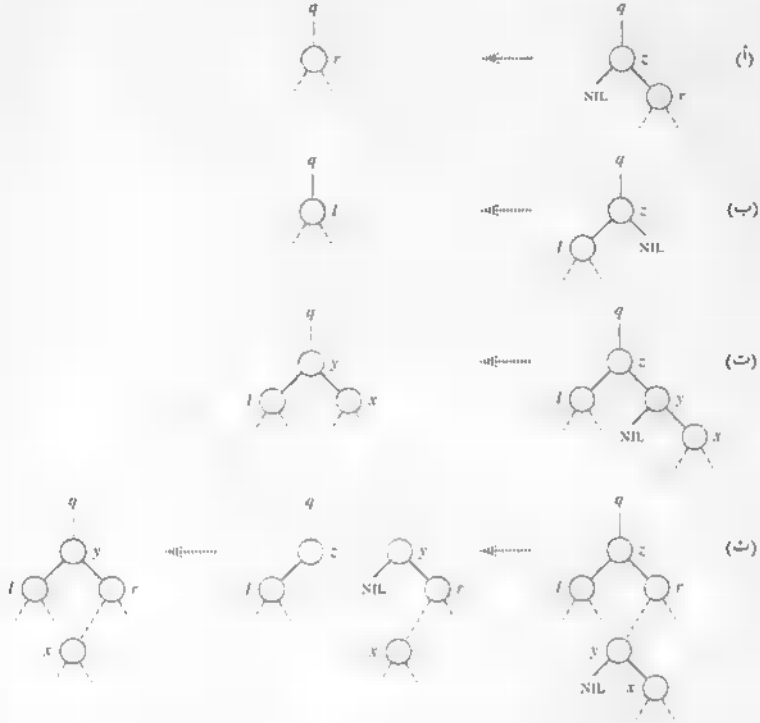
### المحذف

هناك ثلاث حالات أساسية للاستراتيجية الشاملة لحذف عقدة  $z$  من شجرة بحث ثنائية  $T$ ، ولكن إحدى هذه الحالات تطوي على شيء من الدقة والتعقيد، كما سيتبين لنا لاحقاً.



الشكل 3.12 إدراج عنصر مفتاحه 13 في شجرة بحث ثنائية. تشير العقدة الخفيفة التظليل إلى المسار البسيط المنحدر من الجذر باتجاه الموضع الذي جرى فيه إدراج العنصر. يشير الخط المتقطع إلى الوصلة في الشجرة التي أضيفت لإدراج العنصر.

- إذا لم يكن  $z$  أبناء، فإننا ببساطة نزيلها بتغيير أيها بالاستعاضة عن  $z$  بـ  $NIL$  كابن لها.
  - إذا كان  $z$  ابن وحيد، فإننا نرفع هذا الابن ليأخذ مكان  $z$  في الشجرة بتعديل أي  $z$  للاستعاضة عن  $z$  بابن  $z$ .
  - إذا كان  $z$  ابنان، فإننا نوجد  $y$  لاحق  $z$  - الذي يجب أن يكون في الشجرة الفرعية اليمنى لـ  $z$  - ونعمل  $y$  تأخذ مكان  $z$  في الشجرة. ما تبقى من الشجرة الفرعية اليمنى الأصلية لـ  $z$  تصبح الشجرة الفرعية اليمنى الجديدة لـ  $y$ ، وتصبح الشجرة الفرعية اليسرى لـ  $z$  الشجرة الفرعية اليسرى الجديدة لـ  $y$ . ويتمثل وجه التعقيد في هذه الحالة، كما سنرى، في الأهمية للترتبة على كون  $y$  الابن الأيمن لـ  $z$ .
- يأخذ إجراء حذف عقدة معطاة ■ من شجرة بحث ثنائية  $T$  مؤشر  $T$  إلى  $T$  وإلى  $z$  كمحدد  $arguments$ . يُرتَّب الإجراء حالاته بطريقة مختلفة قليلاً عن الحالات الثلاث المذكورة آنفاً، وذلك باعتبار الحالات الأربع المبينة في الشكل 4.12.
- إذا لم يكن  $z$  ابن أيسر (الجزء (أ) من الشكل)، فإننا نستعاض عن ■ بابنه الأيمن، الذي من الممكن أن يكون  $NIL$  أو لا يكون. فإذا كان الابن الأيمن لـ ■ يساوي  $NIL$ ، فإن هذه الحالة تُعالج مع الوضعية التي ليس فيها  $z$  أبناء. أما إذا كان الابن الأيمن لـ  $z$  لا يساوي  $NIL$ ، فإن هذه الحالة تُعالج الوضعية التي فيها  $z$  ابن وحيد، الذي هو ابنه الأيمن.
  - إذا كان  $z$  ابن وحيد، الذي هو ابنه الأيسر (الجزء (ب) من الشكل)، فإننا نستعاض عن  $z$  بابنه الأيسر.
  - وفيما عدا ذلك، فإن  $z$  ابن أيسر وابن أيمن. نوجد  $y$  لاحق  $z$ ، الذي يقع في الشجرة الفرعية اليمنى لـ ■ وليس له ابن أيسر (انظر التمرين 5-2.12). نريد أن نثّر  $y$  من مكانه الحالي ونضعه مكان  $z$  في الشجرة.
- إذا كان  $y$  الابن الأيمن لـ  $z$  (الجزء (ت))، فإننا نستعاض عن  $z$  بـ  $y$ ، تاركين الابن الأيمن لـ  $y$  على حاله.
  - وإلا، يقع  $y$  في الشجرة الفرعية اليمنى لـ  $z$ ، ولكنه ليس الابن الأيمن لـ  $z$  (الجزء (ث)). وفي هذه الحالة، نستعاض بدايةً عن  $y$  بابنه الأيمن، ومن ثم نستعاض عن  $z$  بـ  $y$ .
- ولكي نخزك شجرات فرعية ضمن شجرة بحث ثنائية، فإننا نُعرِّف مساقاً فرعياً "الطعيم" TRANSPLANT، الذي يستعاض عن شجرة فرعية واحدة التي هي ابن لأبيها بشجرة فرعية أخرى. عندما يستعاض TRANSPLANT عن الشجرة الفرعية التي جذرها العقدة  $u$  بالشجرة الفرعية التي جذرها  $v$ ، فإن أبا العقدة  $u$  يصبح أبا العقدة  $v$ ، وينتهي الأمر بأبي  $u$  بحيث يتخذ  $v$  ابناً مناسباً له.



**الشكل 4.12** حذف عقدة  $z$  من شجرة بحث ثنائية. يمكن أن تكون  $z$  هي الجذر، أو أبناً أيسر للعقدة  $q$ ، أو أبناً أيمن لـ  $q$ . (أ) ليس للعقدة  $z$  أي ابن أيسر. نستعيز عن  $z$  بابننا الأيمن  $r$ ، الذي يمكن أن يساوي  $NIL$  أو لا. (ب) للعقدة  $z$  أبناً أيسر  $l$  ولكن ليس لها ابن أيمن. نستعيز عن  $z$  بـ  $l$ . (ت) للعقدة  $z$  ابنان، الابن الأيسر هو العقدة  $l$ ، والابن الأيمن هو لاحقتها  $y$ ، والابن الأيمن لـ  $y$  هو العقدة  $x$ . نستعيز عن  $z$  بـ  $y$ ، ونُحْدِث الابن الأيسر لـ  $y$  ليصبح  $l$  تاركين  $x$  كإبن أيمن لـ  $y$ . (ث) للعقدة  $z$  ابنان ( ابن أيسر  $l$  وابن أيمن  $r$ ) ولاحقتها  $y$  الذي لا يساوي  $r$  يقع في الشجرة الفرعية التي جذرها يقع عند  $r$ . نستعيز عن  $y$  بابنه الأيمن  $x$ ، ونجعل  $y$  أبناً لـ  $r$ . ثم نجعل  $y$  أبناً لـ  $q$  وأبناً لـ  $l$ .

TRANSPLANT( $T, u, v$ )

- 1 if  $u.p == NIL$
- 2      $T.root = v$
- 3 elseif  $u == u.p.left$
- 4      $u.p.left = v$
- 5 else  $u.p.right = v$

6 if  $v \neq \text{NIL}$

7  $v.p = u.p$

يعالج السطران 1 و 2 الحالة التي يكون فيها  $u$  جذر  $T$ . وإلا، فإن  $u$  هو إما ابن أيسر أو ابن أيمن لأبيه. يهتم السطران 3 و 4 بتحديث  $u.p.left$  إذا كان  $u$  ابناً أيسر، وتحديث السطر 5  $u.p.right$  إذا كان  $u$  ابناً أيمن. نسمح بأن يكون  $\equiv$  يساوي NIL، وتحديث السطران 6 و 7  $v.p$  إذا كان  $v$  لا يساوي NIL. لاحظ أن TRANSPLANT لا يحاول تحديث  $v.left$  و  $v.right$ ، إذ تقع مسؤولية القيام بذلك أو عدم القيام به على عاتق مُستدعي TRANSPLANT.

وبوجود الإجراء TRANSPLANT بين أيدينا، نورد فيما يلي الإجراء الذي يحذف عقدة  $z$  من شجرة بحث

ثنائية  $T$ :

TREE-DELETE( $T, z$ )

```

1 if  $z.left == \text{NIL}$ 
2   TRANSPLANT( $T, z, z.right$ )
3 elseif  $z.right == \text{NIL}$ 
4   TRANSPLANT( $T, z, z.left$ )
5 else  $y = \text{TREE-MINIMUM}(z.right)$ 
6   if  $y.p \neq z$ 
7     TRANSPLANT( $T, y, y.right$ )
8      $y.right = z.right$ 
9      $y.right.p = y$ 
10    TRANSPLANT( $T, z, y$ )
11     $y.left = z.left$ 
12     $y.left.p = y$ 
```

يُنفَّذ الإجراء TREE-DELETE الحالات الأربع كما يلي. يُعالج السطران 1-2 الحالة التي ليس فيها  $z$  ابن أيسر، ويُعالج السطران 3-4 الحالة التي يكون فيها  $z$  ابن أيسر وليس له ابن أيمن. يُعالج الأسطر 5-12 الحالتين الباقيتين، والتين  $z$  فيهما ابنان. يوجد السطر 5 العقدة  $y$ ، التي هي لاحق  $z$ . ولما كان  $z$  شجرة فرعية بحى غير فارغة، فيجب أن يكون لاحقه هو العقدة التي لها أصغر مفتاح في تلك الشجرة الفرعية؛ لهذا السبب جرى استدعاء  $\text{TREE-MINIMUM}(z.right)$ . وقد ذكرنا سابقاً أنه ليس لـ  $y$  ابن أيسر. تُرِيد أن نُنزِع  $y$  من مكانه الحالي، وأن نضعه مكان  $z$  في الشجرة. فإذا كان  $y$  الابن الأيمن لـ  $z$ ، فإن الأسطر 10-12 تستعيض عن  $z$  بوصفه ابناً لأبيه  $y$  وتستعيض عن الابن الأيسر لـ  $y$  بالابن الأيسر لـ  $z$ . وإذا لم يكن  $y$  الابن الأيمن لـ  $z$ ، فإن الأسطر 7-9 تستعيض عن  $y$  بوصفه ابناً لأبيه بالابن الأيمن لـ  $y$  وتحول الابن الأيمن لـ  $z$  ليكون الابن الأيمن لـ  $y$ ، ومن ثم تستعيض الأسطر 10-12 عن  $z$  بوصفه ابناً لأبيه  $y$  وتستعيض عن الابن الأيسر لـ  $y$  بالابن الأيسر لـ  $z$ .



يستغرق تنفيذ كل سطر من TREE-DELETE، بما فيها استدعاءات TRANSPLANT زمنًا ثابتًا، ما عدا السطر 5 الذي يستدعي TREE-MINIMUM. ومن ثم، يُنفَّذ TREE-DELETE في زمن  $O(h)$  على شجرة ارتفاعها  $h$ . وبالمجملة، نكون قد أثبتنا للبرهنة التالية.

### مبرهنة 3.12

يمكننا تنجيز عمليات الإدراج والحذف على مجموعة ديناميكية بحيث تُنفَّذ كل منها في زمن  $O(h)$  على شجرة بحث ثنائية ارتفاعها  $h$ . ■

### تمارين

#### 1-3.12

أعطِ نسخة عودية للإجراء TREE-INSERT.

#### 2-3.12

افترض أننا بنينا شجرة بحث ثنائية بإدراج متكرر لقيم متميزة في الشجرة. ناقش أن عدد العقد المختيرة في عملية البحث عن قيمة في الشجرة يساوي عدد العقد المختيرة حين إدخال القيمة أول مرة في الشجرة مضافًا إليه واحد.

#### 3-3.12

يمكننا فرز مجموعة معطاة من  $n$  عددًا بإنشاء شجرة بحث ثنائية أولًا تحتوي هذه الأعداد (باستخدام TREE-INSERT تكرارًا لإدراج الأعداد واحدًا تلو الآخر). ومن ثم طباعة الأعداد باستخدام إجراء تجوال في شجرة وفق الترتيب البيني. ما هو زمن التنفيذ في أسوأ الحالات وفي أحسن الحالات لخوارزمية الفرز هذه؟

#### 4-3.12

هل عملية الحذف "تبديلية" "commutative" بمعنى أن حذف  $x$  ثم  $y$  من شجرة بحث ثنائية يُخلِّف الشجرة نفسها عند حذف  $y$  ثم  $x$ ؟ ناقش لماذا تحصل على الشجرة نفسها أو أعطِ مثالاً معاكسًا يدحض ذلك.

#### 5-3.12

افترض أنه عوضًا من أن تحتفظ العقدة  $x$  بالوصفة  $x.p$ ، التي تشير إلى أبي  $x$ ، فإنها تحتفظ بـ  $x.succ$  التي تشير إلى لاحق  $x$ . أعطِ شبه رماز لتنجيز SEARCH و INSERT و DELETE على شجرة بحث ثنائية  $T$  باستخدام هذا التمثيل. عللنا بأنه يجب أن تُنفَّذ الإجراءات في زمن  $O(h)$ ، حيث  $h$  ارتفاع الشجرة  $T$ . (تلميح: يمكن أن ترغب بتنجيز مساري فرعي بعيد أبا عقدة.)

#### 6-3.12

يمكننا، عندما يكون للعقدة ■ ابنان في TREE-DELETE، اختيار عقدة  $y$  بحيث تكون سابقتها لا لاحقها.

ما هي التعديلات الأخرى اللازم إجراؤها على TREE-DELETE إذا قمنا بذلك. رأى البعض أن استراتيجية عادلة، متمثلة بإعطاء الأولوية نفسها للسابق واللاحق، أي إعطاء الأولوية نفسها للسابق واللاحق، تعطي نتائج تجريبية أفضل. كيف يمكن تغيير TREE-DELETE لإيجاز مثل هذه الاستراتيجية العادلة؟

#### \* 4.12 أشجار بحث ثنائية مبنية عشوائياً

بيننا سابقاً أن كلاً من العمليات الأساسية على شجرة بحث ثنائية تنفذ في زمن  $O(h)$ ، حيث  $h$  ارتفاع الشجرة. على أن ارتفاع شجرة بحث ثنائية يتغير دوماً مع إدراج عناصر وحذفها. على سبيل لمثال، إذا جرى إدراج العناصر  $n$  بترتيب متزايد تمامًا، كانت الشجرة سلسلةً بارتفاع  $n-1$ . من ناحية أخرى، يبين التمرين ب.4-5 أن  $h \geq \lg n$ . وكما في حالة الفرز السريع، يمكننا أن نبين أن سلوك الحالة الوسطى average case أكثر قرباً لأحسن الحالات منه لأسوأ الحالات.

لكننا، ولسوء الحظ، لا نعلم إلا القليل عن الارتفاع الوسطي لشجرة بحث ثنائية عندما يُستخدم الإدراج والحذف مقاً لإنشائها. عندما تُنشأ الشجرة بالإدراج فقط، يكون التحليل قابلاً للتعقب أكثر. لذلك يمكننا أن نُعرف شجرة بحث ثنائية مبنية عشوائياً *randomly built binary search tree* على  $n$  مفتاحاً على أنها شجرة تنشأ من إدراج المفاتيح بترتيب عشوائي في شجرة فارغة في البداية، حيث يكون لكل من تبادل مفاتيح الدخول الـ  $n!$  الاحتمال نفسه. (يطلب اليك في التمرين 3-4.12 أن تبين أن هذا المفهوم يختلف عن افتراض أن كل شجرة بحث ثنائية على  $n$  مفتاحاً لها الاحتمال نفسه.) سنثبت في هذا المقطع المبرهنة التالية.

##### مبرهنة 4.12

القيمة المتوقعة لارتفاع شجرة بحث ثنائية مبنية عشوائياً على  $n$  مفتاحاً تساوي  $O(\lg n)$ ، بافتراض أن جميع المفاتيح متمايزة.

**البرهان** نبدأ بتعريف ثلاثة متحولات عشوائية تساعد في قياس ارتفاع شجرة بحث ثنائية مبنية عشوائياً. نرمز لارتفاع شجرة بحث ثنائية مبنية عشوائياً على  $n$  مفتاحاً بـ  $X_n$ ، ونُعرف **الارتفاع الأسّي** *exponential height*  $Y_n = 2^{X_n}$ . عندما نبني شجرة بحث ثنائية على  $n$  مفتاحاً، فإننا نختار أحد المفاتيح باعتباره جذراً، ونُشر بـ  $R_n$  إلى للتحول العشوائي الذي يمثل مرتبة *rank* هذا المفتاح ضمن مجموعة من  $n$  مفتاحاً أي إن  $R_n$  تمثل للموقع الذي يجب أن يحتله هذا المفتاح إذا قُرِرت مجموعة المفاتيح. يمكن أن تكون قيمة  $R_n$  أيّ عنصر من المجموعة  $\{1, 2, \dots, n\}$ ، باحتمالات متساوية. إذا كان  $R_n = i$ ، فإن الشجرة الفرعية اليسرى للجذر هي شجرة بحث ثنائية مبنية عشوائياً على  $i-1$  مفتاح، والشجرة الفرعية اليمنى للجذر هي شجرة بحث ثنائية مبنية عشوائياً على  $n-i$  مفتاحاً. ولما كان ارتفاع الشجرة الثنائية أكبر بواحد من أكبر

ارتفاعي الشجرتين الفرعيتين للحدذر، فإن الارتفاع الأمسي للشجرة الثنائية يساوي ضعف أكبر ارتفاع أسي للشجرتين الفرعيتين للحدذر. فإذا علمنا أن  $R_n = i$ ، اقتضى ذلك أن

$$Y_n = 2 \cdot \max(Y_{i-1}, Y_{n-i}) .$$

وفي الحالات الأساسية، لدينا  $Y_1 = 1$ ، لأن الارتفاع الأمسي لشجرة بعقدة واحدة يساوي  $2^0 = 1$ ، وللملاءمة نعرف  $Y_0 = 0$ .

نُعرف بعدها متحولات عشوائية مؤشرة  $Z_{n,1}, Z_{n,2}, \dots, Z_{n,n}$  حيث

$$Z_{n,i} = I\{R_n = i\} .$$

ولما كانت قيمة  $R_n$  يمكن أن تكون أي عنصر من المجموعة  $\{1, 2, \dots, n\}$  باحتمالات متساوية، استتبع ذلك أن  $\Pr(R_n = i) = 1/n$  في حالة  $i = 1, 2, \dots, n$ ، ومن ثم، يكون لدينا من التوسطة 1.5

$$E[Z_{n,i}] = 1/n , \quad (1.12)$$

في حالة  $i = 1, 2, \dots, n$ . وبالنظر إلى أن قيمة واحدة بالضبط لـ  $Z_{n,i}$  تساوي 1 وجميع القيم الأخرى تساوي 0، يكون لدينا أيضًا

$$Y_n = \sum_{i=1}^n Z_{n,i} (2 \cdot \max(Y_{i-1}, Y_{n-i})) .$$

وسنبين أن  $E[Y_n] = O(\lg n)$  هو كثير حدود في  $n$ ، وهذا يقتضي أخيرًا أن  $E[X_n] = O(\lg n)$ .

ندعي أن المتحولات العشوائية المؤشرة  $Z_{n,i} = I\{R_n = i\}$  مستقلة عن قيم  $Y_{i-1}$  و  $Y_{n-i}$ . ولأننا اخترنا  $R_n = i$ ، فإن الشجرة الفرعية اليسرى (التي ارتفاعها الأمسي  $Y_{i-1}$ ) تُبنى عشوائيًا على  $i-1$  مفتاحًا التي مراتبها أقل من  $i$ . تشبه هذه الشجرة الفرعية تمامًا أي شجرة بحث ثنائية مبنية عشوائيًا على  $i-1$  مفتاحًا. وباستثناء عدد المفاتيح التي تحتويها بيئة هذه الشجرة الفرعية، فإنها لا تتأثر أبدًا باختيار  $R_n = i$ ، ولذلك فإن المتحولين العشوائيين  $Z_{n,i}$  و  $Y_{i-1}$  مستقلان. وبالمثل، فإن الشجرة الفرعية اليمنى، التي ارتفاعها الأمسي  $Y_{n-i}$ ، مبنية عشوائيًا على  $n-i$  مفتاحًا مراتبها أكبر من  $i$ ، وبنيته مستقلة عن قيمة  $R_n$ ، ومنه فإن المتحولين العشوائيين  $Z_{n,i}$  و  $Y_{n-i}$  مستقلان. إذن يكون لدينا

$$\begin{aligned} E[Y_n] &= E\left[\sum_{i=1}^n Z_{n,i} (2 \cdot \max(Y_{i-1}, Y_{n-i}))\right] \\ &= \sum_{i=1}^n E[Z_{n,i} (2 \cdot \max(Y_{i-1}, Y_{n-i}))] \quad (\text{من خطية التوقع}) \\ &= \sum_{i=1}^n E[Z_{n,i}] E[(2 \cdot \max(Y_{i-1}, Y_{n-i}))] \quad (\text{من الاستقلالية}) \end{aligned}$$

$$= \sum_{i=1}^n \frac{1}{n} E[(2 \cdot \max(Y_{i-1}, Y_{n-i}))] \quad (\text{من المعادلة (1.12)})$$

$$= \frac{2}{n} \sum_{i=1}^n E[(\max(Y_{i-1}, Y_{n-i}))] \quad (\text{من المعادلة (ت. 22)})$$

$$\leq \frac{2}{n} \sum_{i=1}^n (E[Y_{i-1}] + E[Y_{n-i}]) \quad (\text{من التمرين (ت. 3-4)})$$

لما كان كل حد  $E[Y_0], E[Y_1], \dots, E[Y_{n-1}]$  يظهر مرتين في عملية الجمع الأخيرة، مرة في الحد  $E[Y_{i-1}]$  ومرة في الحد  $E[Y_{n-i}]$ ، فإننا نحصل على العلاقة القودية

$$E[Y_n] \leq \frac{4}{n} \sum_{i=0}^{n-1} E[Y_i]. \quad (2.12)$$

وباستعمال طريقة التعويض، سنبيّن أن للعلاقة العودية (2.12) الحل التالي، لجميع الأعداد الصحيحة الموجبة  $n$

$$E[Y_n] \leq \frac{1}{4} \binom{n+3}{3}.$$

وبالقيام بذلك، نستخدم للمطابقة

$$\sum_{i=0}^{n-1} \binom{i+3}{3} = \binom{n+3}{4}. \quad (3.12)$$

(يُطلب إليك في التمرين 1-4.12 برهان هذه للمطابقة.)

نلاحظ، في الحالة الأساسية، أن الحدود  $0 = Y_0 = E[Y_0] \leq \frac{1}{4} \binom{3}{3} = 1/4$  و  $1 = Y_1 = E[Y_1] \leq \frac{1}{4} \binom{1+3}{3} = 1$  تبقى محققة. أما في حالة الاستقراء، فيكون لدينا

$$\begin{aligned} E[Y_n] &\leq \frac{4}{n} \sum_{i=0}^{n-1} E[Y_i] \\ &\leq \frac{4}{n} \sum_{i=0}^{n-1} \frac{1}{4} \binom{i+3}{3} \quad (\text{من الفرضية الاستقرائية}) \\ &= \frac{1}{n} \sum_{i=0}^{n-1} \binom{i+3}{3} \\ &= \frac{1}{n} \binom{n+3}{4} \quad (\text{من المعادلة (3.12)}) \\ &= \frac{1}{n} \cdot \frac{(n+3)!}{4!(n-1)!} \end{aligned}$$

$$\begin{aligned}
 &= \frac{1}{4} \cdot \frac{(n+3)!}{3!n!} \\
 &= \frac{1}{4} \binom{n+3}{3}.
 \end{aligned}$$

لقد حددنا  $E[Y_n]$ ، ولكن هدفنا النهائي هو أن نُحدّد  $E[X_n]$ . وكما يُطلب إليك في التمرين 4-4.12 بيانه، فإن  $f(x) = 2^x$  دالة محدّبة (انظر المقطع ت-3 في الجزء الثاني من هذا الكتاب). لذلك، يمكننا تطبيق متراجحة Jensen's inequality (ت.26)، التي نقول بأن

$$\begin{aligned}
 2^{E[X_n]} &\leq E[2^{X_n}] \\
 &= E[Y_n],
 \end{aligned}$$

كما يلي:

$$\begin{aligned}
 2^{E[X_n]} &\leq \frac{1}{4} \binom{n+3}{3} \\
 &= \frac{1}{4} \cdot \frac{(n+3)(n+2)(n+1)}{6} \\
 &= \frac{n^3 + 6n^2 + 11n + 6}{24}.
 \end{aligned}$$

■ ينتج عن أخذ لogarithم الطرفين أن  $E[X_n] = O(\lg n)$ .

تمارين

1-4.12

أثبت للمعادلة (3.12).

2-4.12

وصف شجرة بحث ثنائية من  $\blacksquare$  عقدة بحيث يكون العمق الوسطي لعقدة في الشجرة يساوي  $\Theta(\lg n)$  في حين يكون ارتفاع الشجرة مساوياً  $\omega(\lg n)$ . أعط الحد الأعلى المقارب لارتفاع شجرة بحث ثنائية من  $n$  عقدة يكون العمق الوسطي لعقدة فيها مساوياً  $\Theta(\lg n)$ .

3-4.12

بيّن أن مفهوم شجرة بحث ثنائية مختارة عشوائياً من  $\blacksquare$  مفتاحاً، حيث احتمالات اختيار كل شجرة من شجرات البحث الثنائية المولفة من  $n$  مفتاحاً متساوية، تختلف عن مفهوم شجرة البحث الثنائية المبينة عشوائياً المشروحة في هذا المقطع. (تلميح: اسرد الاحتمالات الممكنة عندما  $n = 3$ ).

4-4.12

بيّن أن  $f(x) = 2^x$  دالة محدبة.

## \* 5-4.12

ادرس إجراء RANDOMIZED-QUICKSORT يعمل على متتالية من  $n$  عدداً دخلاً متمازاً. برهن، أنه مهما يكن الثابت  $k > 0$ ، فإن جميع تباديل الدخول الـ  $n!$  ما عدا  $O(1/n^k)$  تُنفذ في زمن  $O(n \lg n)$ .

## مسائل

## 1-12 أشجار بحث ثنائية بمفاتيح متساوية

تطرح المفاتيح المتساوية مشكلة عند بناء أشجار بحث ثنائية.

أ. ما هو الأداء المقارب لـ TREE-INSERT عند استخدامها لإدراج  $n$  عنصراً لها المفاتيح نفسها في شجرة بحث ثنائية حالتها الابتدائية فارغة؟

نفترض تحسين TREE-INSERT بإجراء اختياري قبل السطر 5 لمعرفة كون  $z.key = x.key$ ، واختياري قبل السطر 11 لمعرفة كون  $z.key = y.key$ . نطبق إحدى الاستراتيجيات التالية في حال تحققت المساواة. أوجد، من أجل كل استراتيجية، الأداء المقارب لإدراج  $n$  عنصراً بمفاتيح متساوية في شجرة بحث ثنائية حالتها الابتدائية فارغة. (الاستراتيجيات موصوفة لتستخدم في السطر 5، الذي نقارن فيه مفتاح  $z$  بمفتاح  $x$ . استعض عن  $x$  بـ  $y$  للوصول إلى الاستراتيجيات من أجل السطر 11).

ب. احتفظ بالرؤية البوليانية  $b$  عند العقدة  $x$ ، وأسند إلى  $x$  إما  $x.left$  وإما  $x.right$  اعتماداً على قيمة  $x.b$ ، التي تتبدل بين TRUE و FALSE في كل مرة نزور فيها  $x$  خلال عملية إدراج عقدة لها مفتاح  $x$  نفسه.

ت. احتفظ بلائحة العقد التي لها المفاتيح المتساوية عند  $x$ ، وأدرج  $z$  في اللائحة.

ث. أسند إلى  $x$  القيمة  $x.left$  أو  $x.right$  عشوائياً. (أعط الأداء في أسوأ الحالات، واستنبط زمن التنفيذ المتوقع).

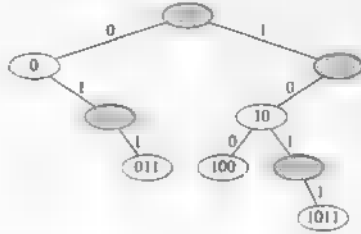
## 2-12 أشجار الأسس

ليكن لدينا سلسلتا المخاريف  $a = a_0 a_1 \dots a_p$  و  $b = b_0 b_1 \dots b_q$ ، حيث كل من  $a_i$  و  $b_j$  هي مجموعة مرتبة من المخاريف. نقول عن سلسلة المخاريف  $a$  إنها أقل أبجدياً من سلسلة المخاريف  $b$  إذا تحقق أحد الشرطين:

1. يوجد عدد صحيح  $z$ ، حيث  $0 \leq z \leq \min(p, q)$ ، وبحيث يكون  $a_i = b_i$  لكل القيم

$$i = 0, 1, \dots, z-1 \text{ أو } a_z < b_z$$

2.  $p < q$  و  $a_i = b_i$  لكل القيم  $i = 0, 1, \dots, p$ .



**الشكل 5.12** تُخزن شجرة أسس radix tree سلسلة محارف البتات 011 و 10 و 100 و 0. يمكننا تحديد مفتاح كل عقدة باحتياز للمسار البسيط من الجذر إلى تلك العقدة. لذلك، لا حاجة إلى تخزين المفاتيح في العقدة؛ تظهر المفاتيح هنا لأغراض توضيحية فقط. تُظلل العقد بشدة حين لا تكون مفاتيحها موجودة في الشجرة؛ والقائمة الوحيدة من وجود هذه العقد إنشاء مسار إلى عقد أخرى ليس غير.

على سبيل المثال، إذا كان  $a$  و  $b$  سلسلي محارف بتات، فإن  $10100 < 10110$  بحسب القاعدة 1 (وبعمل  $j = 3$ )، و  $10100 < 101000$  بحسب القاعدة 2. وهذا الترتيب مشابه للترتيب المُستخدم في معاجم اللغة الإنكليزية.

تُخزن بنية المعطيات من نخت شجرة الأسس radix tree المبينة في الشكل 5.12 سلسلة محارف البتات 011 و 10 و 100 و 0. ولدى البحث عن مفتاح  $a = a_0a_1 \dots a_p$ ، فإننا نذهب مسارًا عند العقدة التي عمقها  $i$  إذا كان  $a_i = 0$ ، وبمينا إذا كان  $a_i = 1$ . لنكن  $S$  مجموعة من سلسلة بتات متمايزة بمجموع أطوالها يساوي  $n$ . بَيِّن كيف نستخدم شجرة الأسس لفرز  $S$  أنجديًا في زمن  $\Theta(n)$ . في حالة المثال في الشكل 5.12، يجب أن تكون نتيجة الفرز هي المتتالية 0, 011, 10, 100, 1011.

### 3-12 العمق الوسطي لعقدة في شجرة بحث ثنائية مبنية عشوائيًا

نبرهن في هذه المسألة أن العمق الوسطي لعقدة في شجرة بحث ثنائية مبنية عشوائيًا على  $n$  عقدة يساوي  $O(\lg n)$ . ومع أن هذه النتيجة أضعف من نتيجة المبرهنة 4.12، فإن الأسلوب الذي سنستخدمه يُظهر تشابهًا مدهشًا بين بناء شجرة بحث ثنائية وتنفيذ الإجراء RANDOMIZED-QUICKSORT من المقطع 3.7.

تُعرف الطول الكلي للمسار  $P(T)$  total path length لشجرة ثنائية  $T$  على أنه مجموع عمق كل العقد  $x$  في الشجرة  $T$ ، ونرمز له بـ  $d(x, T)$ .

أ. برهن أن العمق الوسطي لعقدة في  $T$  يساوي

$$\frac{1}{n} \sum_{x \in T} d(x, T) = \frac{1}{n} P(T) .$$

وهكذا تُثبت أن القيمة المتوقعة لـ  $P(T)$  تساوي  $O(n \lg n)$ .

ب. نُشر بـ  $T_L$  و  $T_R$  إلى الشجرة الفرعية اليسرى والشجرة الفرعية اليمنى للشجرة  $T$  على الترتيب. يبرهن أنه إذا كانت الشجرة  $T$  ذات  $n$  عقدة، فإن

$$P(T) = P(T_L) + P(T_R) + n - 1 .$$

ت. نُشر بـ  $P(n)$  إلى وسطي الطول الكلي للمسار لشجرة بحث ثنائية مبنية عشوائيًا على  $n$  عقدة، يبرهن أن

$$P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n-i-1) + n-1) .$$

ث. يبرهن كيف يمكن إعادة كتابة  $P(n)$  كما يلي

$$P(n) = \frac{2}{n} \sum_{k=1}^{n-1} P(k) + \Theta(n) .$$

ج. استنتج، بالعودة إلى التحليل البديل للنسخة ذات العشوائية للفرز السريع quicksort المعطاة في المسألة 3-7، أن  $P(n) = O(n \lg n)$ .

نختار، عند كل استدعاء عودي لـ quicksort، عنصرًا محوريًا عشوائيًا لتجزئة مجموعة العناصر الخاضعة للفرز. نُجرِّئ كلَّ عقدة في شجرة البحث الثنائية مجموعة العناصر التي تقع في شجرة فرعية جذورها تلك العقدة.

و. وصِّف تحجُّرًا للفرز السريع quicksort تكون فيه المقارنات المستخدمة لفرز مجموعة من العناصر هي نفسها المقارنات المستخدمة لإدراج عناصر في شجرة بحث ثنائية. (قد يختلف ترتيب هذه المقارنات، ولكن يجب إجراء المقارنات نفسها.)

#### 4-12 عدد الأشجار الثنائية المختلفة

نُشر  $b_n$  إلى عدد الأشجار المختلفة من  $n$  عقدة. سنكتشف في هذه المسألة صيغة لـ  $b_n$ ، إضافة إلى التقدير للمقارب.

أ. يبرهن أن  $b_0 = 1$  وأنه في حال  $n \geq 1$  فإن

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-1-k} .$$

ب. بالعودة إلى المسألة 4-4 لتعريف الدالة المُولَّدة generating function. لتكن  $B(x)$  الدالة المُولَّدة

$$B(x) = \sum_{n=0}^{\infty} b_n x^n .$$



بَيِّنْ أن  $B(x) = xB(x)^2 + 1$ ، وبذلك تكون إحدى الطرق للتعبير عن  $B(x)$  بصيغة مغلقة  
في closed form

$$B(x) = \frac{1}{2x} (1 - \sqrt{1 - 4x}) .$$

يُعطى نشر تايلور Taylor expansion لـ  $f(x)$  حول النقطة  $x = a$  بالعلاقة

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!} (x-a)^k ,$$

حيث  $f^{(k)}(x)$  هو المشتق من المرتبة  $k$  لـ  $f$  محسوبًا عند النقطة  $x$ .

ت. بَيِّنْ أن

$$b_n = \frac{1}{n+1} \binom{2n}{n}$$

(عدد كاتالان Catalan number من المرتبة  $n$ ) باستخدام نشر تايلور لـ  $\sqrt{1-4x}$  حول النقطة  $x=0$ . (يمكنك أن تستعمل - بدلاً من نشر تايلور - تعميم نشر ثنائي الحد (ت.4) لأش غير صحيح  $n$ ، حيث يمكننا كتابة  $\binom{n}{k}$  لأي قيمة حقيقية  $n$  ولأي عدد صحيح  $k$  على النحو الآتي  $n(n-1)\dots(n-k+1)/k!$  حين تكون  $k \geq 0$ ، و 0 فيما عدا ذلك.)

ث. بَيِّنْ أن

$$b_n = \frac{4^n}{\sqrt{\pi} n^{3/2}} (1 + O(1/n)) .$$

## ملاحظات الفصل

يحتوي Knuth [211] دراسة جيدة لأشجار البحث الثنائية البسيطة إضافة إلى أشجار متنوعة عديدة. ويبدو أن اكتشاف أشجار البحث الثنائية قد تمَّ إفراديًا على يد عدد من الأشخاص في أواخر خمسينيات القرن العشرين. وكثيرًا ما تُسمى أشجار الأسس "tries"، وقد صيغت هذه الكلمة من الحروف الوسطى لكلمة retrieval. وهي أيضًا مدروسة في Knuth [211].

يحتوي كثيرٌ من النصوص، ومنها أول إصدارين لهذا الكتاب، طريقةً بسيطةً نوعًا ما لخوارزمية حذف عقدة من شجرة بحث ثنائية إذا كان كلا ابنها موجودين. فبدلاً من الاستعاضة عن العقدة  $z$  بلحقها  $y$ ، فإننا نحذف العقدة  $y$  ولكن مع نسخ مفتاحها وللعطيات التابعة لها في العقدة  $z$ . ويتمثل الجانب الملبّي لهذا النهج في أن العقدة المحذوفة فعليًا قد لا تكون هي العقدة المُحررة إلى إجراء الحذف. وإذا احتفظت مكونات

أعزى للبرنامج بمؤشرات إلى عقد في الشجرة، فلربما انتهى الأمر خطأ إلى مؤشرات "بالية" إلى عقد جرى حذفها. ومع أن طريقة الحذف المقدمة في هذا الإصدار من هذا الكتاب أعقد قليلاً، فإنها تضمن أن استدعاء لحذف عقدة  $z$  سيحذف العقدة  $z$  فقط وليس غير.

يبيّن المقطع 5.15 كيف نبنى شجرة بحث ثنائية أمثلية إذا عرفنا تواترات البحث قبل إنشاء الشجرة. وهذا يعني، أننا إذا علمنا تواتر البحث عن كل مفتاح وتواتر البحث عن القيم التي تقع بين المفاتيح في الشجرة، أمكننا بناء شجرة بحث ثنائية تقوم بمقتضاها بمجموعة للبحث تخضع لهذه التواترات بدراسة أقل عدد من العقد. يعود الفضل في البرهان الوارد في المقطع 4.12، الذي يحدّ الارتفاع المتوقع لشجرة بحث ثنائية مبنية عشوائياً إلى Aslam [24]. كذلك يُقدم Martinez و Roura [243] حواريات ذات عشوائية مضافة بغية الإدراج في شجرة بحث ثنائية والحذف منها، والتي تكون فيها نتيجة كل من العمليتين شجرة بحث ثنائية عشوائية. غير أن تعريفهما لشجرة بحث ثنائية عشوائية يختلف اختلافاً طفيفاً عن تعريف شجرة بحث ثنائية مبنية عشوائياً، الوارد في هذا الفصل.

## 13 الأشجار الحمراء-السوداء

رأينا في الفصل 12 أنه يمكن لشجرة بحث ثنائية ذات ارتفاع  $h$  أن تدعم أياً من عمليات المجموعات الديناميكية الأساسية - مثل البحث SEARCH والثابت PREDECESSOR والحذف SUCCESSOR والحد الأدنى MINIMUM والحد الأعلى MAXIMUM والإدراج INSERT والحذف DELETE - في زمن  $O(h)$ . لذلك، تكون عمليات المجموعات سريعة إذا كان ارتفاع شجرة البحث صغيراً. فإذا كان ارتفاعها كبيراً فقد لا تُنفَّذ هذه العمليات بسرعة أكبر مما لو استُخدمت قائمة مرتبطة. والأشجار الحمراء-السوداء هي أحد أشكال كثيرة لأشجار البحث التي جعلت "متوازنة" لكي تضمن أن تستغرق عمليات المجموعات الديناميكية الأساسية في أسوأ الحالات زمناً  $O(\lg n)$ .

### 1.13 خصائص الأشجار الحمراء-السوداء

**الشجرة الحمراء-السوداء** *red-black tree* هي شجرة بحث ثنائية تتضمن بت تخزين إضافي واحد في كل عقدة من عقدها، يعبّر عن لون العقدة الذي يمكن أن يكون أحمر أو أسود. ويتقيد ألوان العقد على أي مسار بسيط، من الجذر إلى إحدى الأوراق، تتضمن الأشجار الحمراء-السوداء أنه لا يوجد مسار يتجاوز طوله ضعف طول أي مسار آخر، أي إن الشجرة متوازنة *balanced* تقريباً.

أصبحت كل عقدة من الشجرة تتضمن الوصفات *attributes* التالية: اللون *color* والمفتاح *key* والابن الأيمن *right* والابن الأيسر *left* والأب *p*. فإذا لم يكن الأب أو أحد الأبناء موجوداً تكون قيمة حقل المؤشر الموافق للعقدة معلومة *NIL*. وسننظر إلى هذه المؤشرات المعلومه (*NIL*) على أنها مؤشرات إلى أوراق (عقد خارجية) في الشجرة الثنائية، وإلى العقد العادية الحاملة للمفاتيح على أنها عقد داخلية في الشجرة.

الشجرة الحمراء-السوداء هي شجرة بحث ثنائية تحقق الخصائص الحمراء-السوداء *red-black*

*properties* التالية:

1. كل عقدة إما أن تكون حمراء أو سوداء.

2. عقدة الجذر سوداء.

3. كل ورقة (NIL) هي سوداء.
4. إذا كانت إحدى العقد حمراء، كانت عقدتا أبنائها سوداويتين.
5. كل المسارات البسيطة من أي عقدة إلى الأوراق النازلة منها تحتوي العدد نفسه من العقد السوداء.

يُظهر الشكل 1.13 (أ) مثلاً على شجرة حمراء-سوداء.

نستخدم حارساً وحيداً لتمثيل القيم المعدومة NIL لتسهيل التعامل مع الشروط الحديثة في رماز الأشجار الحمراء-السوداء (انظر الصفحة 239). ففي شجرة حمراء-سوداء  $T$  يكون الحارس  $T.nil$  غرضاً له نفس واصفات أي عقدة عادية في الشجرة. ويكون واصف اللون  $color$  فيه أسود BLACK، وأما بقية الحقول  $-$  الأب  $p$  والابن الأيسر  $left$  والابن الأيمن  $right$  والفتاح  $key$  - فيمكن أن تأخذ قيماً اعتباطية. وكما يُظهر الشكل 1.13 (ب) فإن كل المؤشرات إلى NIL يستعاض عنها بمؤشرات إلى الحارس  $T.nil$ .

نستخدم الحارس بحيث يمكننا معالجة ابن معدوم NIL لعقدة ما  $x$  كمقدمة عادية أبوها  $x$ . ومع أنه يمكننا بدل ذلك أن نضيف عقدة حارس متمايزة لكل عقدة معلومة NIL في الشجرة، وبحيث يكون الأب الخاص بكل قيمة معدومة NIL معرفاً تعريفًا جيدًا، فإن ذلك النهج سيبدد المحجوم. بدلاً من ذلك نستعبد الحارس الوحيد  $T.nil$  لتمثيل جميع القيم للمعدومة NIL - أي كل الأوراق والأب الخاص بالجذر. إن قيم واصفات الحارس  $p$  و  $left$  و  $right$  و  $key$  ليست ذات أهمية، ومع ذلك يمكننا - للتسهيل - أن نعطيهما قيماً اصطلاحية خلال سير إجراء معين.

نوجه اهتمامنا عمومًا إلى العقد الداخلية من الشجرة الحمراء-السوداء لأنها تحمل قيم المفاتيح. سنُغفل فيما تبقى من هذا الفصل الأوراق عند رسم شجرة حمراء-سوداء، كما يبين الشكل 1.13 (ت).

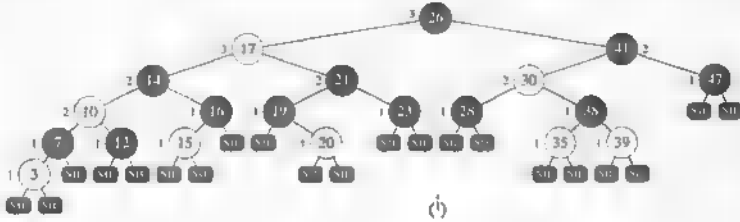
نسمي عدد العقد السوداء على أي مسار بسيط من عقدة ما  $x$  (غير متضمنة) نزولاً إلى ورقة ما الارتفاع الأسود  $black-height$  للعقدة، ونرمز له  $bh(x)$ . وحسب الخاصية رقم 5، فإن مفهوم الارتفاع الأسود معرّف جيداً، لأن كل المسارات البسيطة النازلة من العقدة لها العدد نفسه من العقد السوداء. نعرّف الارتفاع الأسود لشجرة حمراء-سوداء على أنه الارتفاع الأسود لجذرها. تُظهر التوطئة التالية السبب في أن الأشجار الحمراء-السوداء أشجار بحث جيدة.

### توطئة 1.13

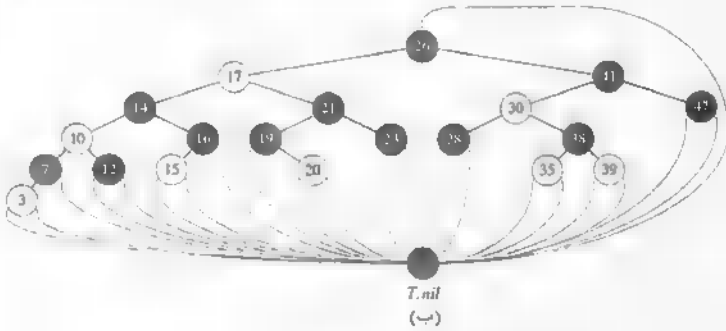
كل شجرة حمراء-سوداء ذات  $n$  عقدة داخلية يكون ارتفاعها على الأكثر  $2 \lg(n+1)$

**البرهان** نبدأ ببيان أن الشجرة الفرعية التي جذرها عند أية عقدة  $x$  تتضمن على الأقل  $2^{bh(x)} - 1$  عقدة داخلية. نرهن على هذا بالاستقراء على ارتفاع  $x$ . إذا كان ارتفاع  $x$  هو 0، فإن  $x$  يجب أن يكون ورقة ( $T.nil$ )، وتكون الشجرة الفرعية التي جذرها  $x$  تتضمن فعلياً  $0 = 2^0 - 1 = 2^{bh(x)} - 1$  عقدة داخلية

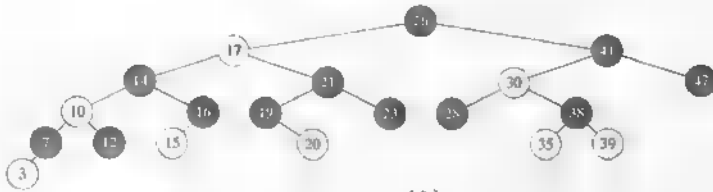
على الأقل. نفترض خطوة الاستقرار عقدة ما  $x$  ذات ارتفاع موجب، وهي عقدة داخلية لها ابنان. كل ابن له ارتفاع أسود مقداره  $bh(x)$  أو  $bh(x)-1$ ، وذلك تبعاً للونه (أحمر أو أسود على الترتيب). ولما كان ارتفاع



(أ)



(ب)



(ت)

**الشكل 1.13** شجرة حمراء-سوداء تظهر فيها العقد السوداء بلون غامق والحمراء مظللة. كل عقدة في الشجرة الحمراء-السوداء إما أن تكون حمراء أو سوداء، وإما أي عقدة حمراء سوداوان، وكل مسار بسيط من عقدة ما إلى الأوراق النازلة منها يحتوي العدد نفسه من العقد السوداء. (أ) كل ورقة تظهر بأنها معدومة NIL تكون سوداء. كل عقدة غير معدومة تكون معلّمة بارتفاعها الأسود والارتفاع الأسود للعقد المعلوم هو 0. (ب) الشجرة الحمراء-السوداء نفسها لكن مع الاستعاضة عن كل عقدة معدومة NIL بالحارس الوحيد  $T.nil$  الذي يكون أسود دوماً، ومع إغفال الارتفاعات السوداء. العقدة الأب للجذر هي الحارس كذلك. (ت) الشجرة الحمراء-السوداء نفسها لكن مع إغفال الأوراق والأب الخاص بالجذر كلياً. نستخدم هذا الأسلوب في الرسم حتى غاية هذا الفصل.

أحد أبناء  $x$  أقل من ارتفاع  $x$  نفسها، فيمكننا تطبيق الفرضية الاستقرائية لنستنتج أنَّ كل ابن لديه على الأقل  $2^{bh(x)-1} - 1$  عقدة داخلية. إذن فالشجرة الفرعية ذات الجذر  $x$  تتضمن على الأقل  $2^{bh(x)} - 1 = 2^{bh(x)-1} + 1 + (2^{bh(x)-1} - 1) + (2^{bh(x)-1} - 1)$  عقدة داخلية، وهو المطلوب.

ولاستكمال برهان التوطئة، ليكن  $\blacksquare$  ارتفاع الشجرة. فبحسب الخاصية 4، يجب أن يكون على الأقل نصف العقد الواقعة على أي مسار بسيط من الجذر إلى الأوراق، باستثناء الجذر، سوداء. وهذا يستتبع أن الارتفاع الأسود للجذر يجب أن يكون على الأقل  $h/2$ ، ومن ثمَّ

$$n \geq 2^{h/2} - 1.$$

بنقل الواحد إلى الجانب الأيسر، وأخذ لغارتم الطرفين يكون لدينا  $\lg(n+1) \geq h/2$  أو  $h \leq 2\lg(n+1)$ . ■

ينتج مباشرةً عن هذه التوطئة أننا يمكن أن ننجز عمليات المجموعات الديناميكية: SEARCH، PREDECESSOR و SUCCESSOR، MAXIMUM، MINIMUM في زمن  $O(\lg n)$  على الأشجار الحمراء-السوداء، لأن كلاً منها يمكن أن تُنفَّذ في زمن  $O(h)$  على شجرة بحث ثنائية ذات ارتفاع  $h$  (كما تقدّم في الفصل 12)، وأي شجرة حمراء-سوداء ذات  $n$  عقدة هي شجرة بحث ثنائية بارتفاع  $O(\lg n)$ . (يجب طبعاً أن تُستبدل بكل مواضع ورود NIL في خوارزميات الفصل 12 كلمة T.nil). ومع أنَّ الخوارزميات TREE-DELETE و TREE-INSERT في الفصل 12 تنفَّذ في زمن  $O(\lg n)$  عندما يكون الدخّل شجرة حمراء-سوداء، إلا أنها لا تدعم مباشرةً عمليات المجموعات الديناميكية: INSERT و DELETE لأنها لا تضمن أن تكون شجرة البحث الثنائية للمعدّلة شجرة حمراء-سوداء. مع ذلك، سنرى في المقطع 3.13 والمقطع 4.13 كيف يمكننا تنفيذ هاتين العمليتين في زمن  $O(\lg n)$ .

## تمارين

### 1-1.13

على غرار الشكل 1.13(أ)، ارسم شجرة البحث الثنائية الكاملة ذات الارتفاع 3، والمفاتيح  $\{1, 2, \dots, 15\}$ . أضف الأوراق NIL ولوّّن العقد بثلاث طرق مختلفة، بحيث تكون الارتفاعات السوداء للأشجار الحمراء-السوداء الناتجة هي 2 و 3 و 4.

### 2-1.13

ارسم الشجرة الحمراء-السوداء التي تنتج بعد استدعاء TREE-INSERT على الشجرة التي في الشكل 1.13 مع المفتاح 36. وإذا كانت العقدة المدرجة ملوّنة بالأحمر، فهل الشجرة الناتجة هي شجرة حمراء-سوداء؟ ماذا لو كانت ملوّنة بالأسود؟

## 3-1.13

لتعرف شجرة حمراء-سوداء مرتخلة *relaxed red-black tree* على أنها شجرة بحث ثنائي تحقق خصائص الأشجار الحمراء-السوداء 1 و 3 و 4 و 5. وبشعر آخر، يمكن أن يكون الجذر أحمر أو أسود. لناخذ شجرة  $T$  من هذا النوع جذرها أحمر. إذا لَوَّنا جذر  $T$  بالأسود ولم نجر أي تغيير آخر على  $T$ ، فهل تكون الشجرة الناتجة شجرة حمراء-سوداء؟

## 4-1.13

افترض أننا "نقتص" كل عقدة حمراء في شجرة حمراء-سوداء في أمها السوداء، بحيث يصبح أبناء العقدة الحمراء أبناء للعقدة الأم السوداء. (تجاهل ما يحدث للمفاتيح.) ما هي الدرجات الممكنة لعقدة سوداء بعد أن يجري امتصاص جميع أبنائها الحمراء؟ ماذا تقول عن أعماق الأوراق في الشجرة الناتجة؟

## 5-1.13

بين أن أطول مسار بسيط من عقدة ■ في شجرة حمراء-سوداء إلى ورقة نازلة منها، يبلغ طوله على الأكثر ضعف طول أقصر مسار بسيط من عقدة  $x$  إلى ورقة نازلة منها.

## 6-1.13

ما هو العدد الأعظمي المحتمل للعقد الداخلية في شجرة حمراء-سوداء ارتفاعها الأسود  $k$ ؟ وما هو العدد الأصغري المحتمل؟

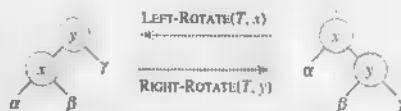
## 7-1.13

صيف شجرة حمراء-سوداء تتضمن  $n$  مفتاح تكون فيها نسبة العقد الداخلية الحمراء إلى العقد الداخلية السوداء أعلى ما يمكن. وما هي هذه النسبة؟ ما هي الشجرة التي يكون لها أدنى نسبة محتملة؟ وما هي هذه النسبة؟

## 2.13 الدورانات

تستغرق عمليات أشجار البحث  $TREE-INSERT$  و  $TREE-DELETE$  زمناً  $O(\lg n)$  عند تنفيذها على شجرة حمراء-سوداء تتضمن  $n$  مفتاحاً. ولما كانت هذه العمليات تعمل الشجرة فإن النتيجة يمكن أن تفرق خصائص الأشجار الحمراء-السوداء المذكورة في المقطع 1.13. ولإعادة تحقيق هذه الخصائص، يجب أن نغير ألوان بعض العقد في الشجرة وأن نغير بنية المؤشر.

نغير بنية المؤشر بالدورانات *rotation*، وهي عملية محلية في شجرة بحث تحافظ على خاصية شجرة البحث النهائية. يُظهر الشكل 2.13 نوعي الدورانات: الدورانات إلى اليسار والدورانات إلى اليمين. عندما نُجري دوراتاً إلى اليسار على عقدة  $x$ ، نفترض أن ابنها الأيمن  $y$  ليس معدوماً  $T.nil$ ؛ ويمكن أن تكون  $x$  أية عقدة في



**الشكل 2.13** عمليات الدوران في شجرة بحث ثنائية. تحول العملية  $LEFT-ROTATE(T, x)$  تشكيلة العقدين الموجودة إلى يمين الشكل إلى التشكيلة الموجودة إلى يسار الشكل بتغير عدد ثابت من المؤشرات. تحول العملية المعاكسة  $RIGHT-ROTATE(T, y)$  التشكيلة الموجودة إلى اليسار إلى التشكيلة الموجودة إلى اليمين. وتمثل الأحرف  $\alpha$  و  $\beta$  و  $\gamma$  أشجاراً فرعية اعتباطية. تحفظ عملية الدوران خاصية شجرة البحث الثنائية: المفاتيح في  $\alpha$  تسبق  $x$ ,  $key$ , التي تسبق المفاتيح في  $\beta$ , التي تسبق  $y$ , التي تسبق بدورها المفاتيح في  $\gamma$ .

الشجرة ابنها الأيمن غير معدوم  $T.nil$ . الدوران الأيسر "يرتكز" على الرابط من  $x$  إلى  $y$ . ويجعل من  $y$  الجذر الجديد للشجرة الفرعية، ويصبح  $x$  الابن الأيسر لـ  $y$ ، ويصبح الابن الأيسر لـ  $y$  هو الابن الأيمن لـ  $x$ . نفترض شبه رماز  $LEFT-ROTATE$  أن  $x.right \neq T.nil$  وأن أبا الجذر هو  $T.nil$ .

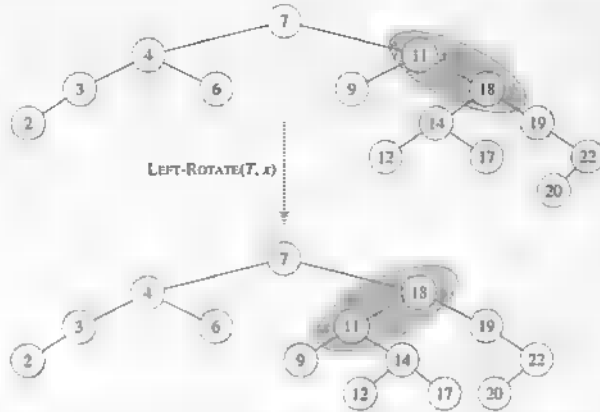
```

LEFT-ROTATE(T, x)
1  y = x.right           // set y
2  x.right = y.left // turn y's left subtree into x's right subtree
3  if y.left != T.nil
4      y.left.p = x
5  y.p = x.p             // link x's parent to y
6  if x.p == T.nil
7      T.root = y
8  elseif x == x.p.left
9      x.p.left = y
10 else x.p.right = y
11 y.left = x             // put x on y's left
12 x.p = y

```

يُظهر الشكل 3.13 مثالاً عن كيفية تعديل الإجراء  $LEFT-ROTATE$  شجرة بحث ثنائية. رماز  $RIGHT-ROTATE$  هو نظير للرماز السابق. يعمل كل من  $LEFT-ROTATE$  و  $RIGHT-ROTATE$  في زمن  $O(1)$ . المؤشرات فقط هي التي تتغير في الدوران؛ وتبقى الواصفات الأخرى في العقدة كما هي.





**الشكل 3.13** مثال عن كيفية تعديل الإجراء  $LEFT-ROTATE(T, x)$  شجرة بحث ثنائية. ينتج عن المسير بالترتيب الداخلي في كل من شجرة الدخل والشجرة المعدلة القائمة نفسها من المفاتيح.

**تمارين**

**1-2.13**

اكتب شبه رماز العملية  $RIGHT-ROTATE$ .

**2-2.13**

برهن أن في كل شجرة بحث ثنائية ذات  $n$  عقدة  $n - 1$  دوراناً يمكننا تماماً.

**3-2.13**

لتكن  $a$  و  $b$  و  $c$  عقداً اعتباطية في الأشجار الفرعية  $\alpha$  و  $\beta$  و  $\gamma$  على الترتيب في الشجرة اليمنى من الشكل 2.13. كيف يتغير عمق كل من  $a$  و  $b$  و  $c$  عند إجراء دوران نحو اليسار على العقدة  $x$  المبينة في الشكل؟

**4-2.13**

يُبين أن أي شجرة بحث ثنائية اعتباطية ذات  $n$  عقدة يمكن أن تُحوّل إلى أية شجرة بحث ثنائية اعتباطية أخرى ذات  $n$  عقدة باستخدام  $O(n)$  دوراناً. (تلميح: يُبين أولاً أنه يكفي  $n - 1$  دوراناً إلى اليمين على الأكثر لتحويل الشجرة إلى سلسلة تبدأ من اليمين.)

**\* 5-2.13**

نقول إن شجرة بحث ثنائية  $T_1$  يمكن أن تُحوّل يميناً *right-converted* إلى شجرة بحث ثنائية  $T_2$  إذا أمكن الحصول على  $T_2$  من  $T_1$  عبر سلسلة من استدعاءات  $RIGHT-ROTATE$ . أعط مثلاً عن شجري  $T_1$  و  $T_2$  بحيث لا يمكن لـ  $T_1$  أن تُحوّل يميناً إلى  $T_2$ . ثم يُبين أنه إذا أمكن تحويل شجرة  $T_1$  يميناً إلى  $T_2$ ، فإن هذا التحويل يمكن أن يجري باستخدام  $O(n^2)$  استدعاء لـ  $RIGHT-ROTATE$ .

## 3.13 الإدراج

يمكننا إدراج عقدة في شجرة حمراء-سوداء ذات  $n$  عقدة في زمن  $O(\lg n)$ . ولإجراء ذلك نستخدم نسخة معدلة قليلاً من إجراء TREE-INSERT (للقطع 3.12) لإدراج عقدة  $z$  في شجرة  $T$  كما لو كانت شجرة بحث ثنائية عادية، ثم نُلوّن  $z$  بالأحمر. (يُطلب إليك في التمرين 1-3.13 أن تفسّر في اختيارنا للوّن العقدة  $z$  بالأحمر بدلاً عن الأسود.) ولضمان الحفاظ على الخصائص الحمراء-السوداء، ن استدعي إجراء مساعدًا RB-INSERT-FIXUP لإعادة لوّن العقد وإجراء الدورانات. إنَّ استدعاء RB-INSERT( $T, z$ ) يدرج عقدة  $z$ ، يُفترض أن مفتاحها قد مُلئ سابقاً، ضمن الشجرة الحمراء-السوداء  $T$ .

RB-INSERT( $T, z$ )

```

1   $y = T.nil$ 
2   $x = T.root$ 
3  while  $x \neq T.nil$ 
4       $y = x$ 
5      if  $z.key < x.key$ 
6           $x = x.left$ 
7      else  $x = x.right$ 
8   $z.p = y$ 
9  if  $y == T.nil$ 
10      $T.root = z$ 
11  elseif  $z.key < y.key$ 
12      $y.left = z$ 
13  else  $y.right = z$ 
14   $z.left = T.nil$ 
15   $z.right = T.nil$ 
16   $z.color = RED$ 
17  RB-INSERT-FIXUP( $T, z$ )

```

هناك أربعة اختلافات بين الإجراءين TREE-INSERT و RB-INSERT. أولاً، كل القيم  $nil$  في TREE-INSERT جرى الاستعاضة عنها بـ  $T.nil$ . ثانياً، نجعل قيمة كل من  $z.left$  و  $z.right$  مساوية  $T.nil$  في الأسطر 14-15 من RB-INSERT، للحفاظ على بنية الشجرة. ثالثاً، نُلوّن  $z$  بالأحمر في السطر 16. رابعاً، لما كان لوّن  $z$  بالأحمر يمكن أن يسبب خرقاً لإحدى الخصائص الحمراء-السوداء، فإننا ن استدعي RB-INSERT-FIXUP( $T, z$ ) في السطر 17 من RB-INSERT لاستعادة الخصائص الحمراء-السوداء.

RB-INSERT-FIXUP( $T, z$ )

```

1  while  $z.p.color == RED$ 
2      if  $z.p == z.p.p.left$ 
3           $y = z.p.p.right$ 
4          if  $y.color == RED$ 

```

```

5      z.p.color = BLACK // Case 1
6      y.color = BLACK // Case 1
7      z.p.p.color = RED // Case 1
8      z = z.p // Case 1
9      else if z == z.p.right
10     z = z.p // Case 2
11     LEFT-ROTATE(T,z) // Case 2
12     z.p.color = BLACK // Case 3
13     z.p.p.color = RED // Case 3
14     RIGHT-ROTATE(T,z.p.p) // Case 3
15     else (same as then clause with "right" and "left" exchanged)
16     T.root.color = BLACK

```

ولمعرفة كيفية عمل الإجراء RB-INSERT-FIXUP سنحوّي دراستنا للرمز إلى ثلاث خطوات رئيسية. أولاً، سَنُحدّد ما هي الحروفات للخصائص الحمراء-السوداء التي حصلت في RB-INSERT عند إدراج العقدة  $z$  وتلوينها بالأحمر. ثانياً، سنفحص الهدف العام لخطة **while** في الأسطر 1-15. أخيراً، سنستعرض كلاً من الحالات الثلاث<sup>1</sup> ضمن جسم حلقة **while** ونرى كيف حققت الهدف. يُظهر الشكل 4.13 كيف يعمل RB-INSERT-FIXUP على عينة شجرة حمراء-سوداء.

أيّ من الخصائص الحمراء-السوداء يمكن أن تُخرق بعد استدعاء RB-INSERT-FIXUP؟ الخاصية 1 تبقى صحيحة بالتأكيد، وكذلك الخاصية 3، لأن كلا ابني العقدة الحمراء الجديدة المدرجة حديثاً هما الحارس  $T.nil$ . أما الخاصية 5 التي تعني أنّ عدد العقدة السوداء هو نفسه في جميع انتمارات البسيطة من عقدة معينة، هي أيضاً محققة، لأن العقدة  $z$  تستبدل الحارس (الأسود)، والعقدة  $z$  حمراء ولديها أبناء حراس. لذلك فإنّ الخاصيتين اللتين يمكن أن يجري خرقهما هما الخاصية 2 التي تتطلب أن يكون الجذر أسود، والخاصية 4 التي تعني أنّه لا يمكن لعقدة حمراء أن يكون لها ابن أحمر. وكلا الخرفتين ينتجان عن تلوين  $z$  بالأحمر. تُخرق الخاصية 2 إذا كان  $z$  هو الجذر، وتُخرق الخاصية 4 إذا كان أبو  $z$  أحمر. يُظهر الشكل 4.13(أ) خرق الخاصية 4 بعد إدراج العقدة  $z$ .

تُحفظ حلقة **while** في الأسطر 15-16 باللامتغير الثلاثي الأجزاء الآتي في بداية كل تكرار من الحلقة:

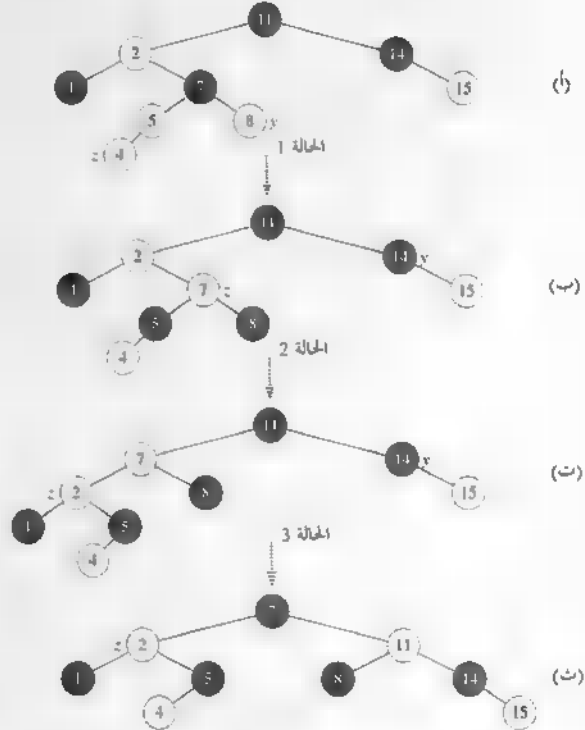
أ. العقدة  $z$  حمراء.

ب. إذا كان  $z.p$  هو الجذر، فإنه يكون أسود.

ت. إذا خُرقت الشجرة أيّاً من الخصائص الحمراء-السوداء، فستخرق واحدة منها على الأكثر، وسيكون الخرق إما للخاصية 2 أو الخاصية 4. فإذا خُرقت الشجرة الخاصية 2 فسبب ذلك أن  $z$  هو الجذر وهو أحمر. وإذا خُرقت الشجرة الخاصية 4، فسبب ذلك أن لون كلٍّ من  $z$  و  $z.p$  أحمر.

<sup>1</sup> الحالة 2 تنتهي إلى الحالة 3، أي أنّ حاتين الحاتين لا تقصي إحداهما الأخرى.

الجزء (ت) الذي يتناول خرق الخصائص الحمراء-السوداء، هو الجزء الأكثر أهمية لبيان استعادة هذه الخصائص في RB-INSERT-FIXUP والذي نستخدمه دوماً لفهم الحالات ضمن الرماز. ولما كنا نركز على العقدة  $z$  والعقد التي بجوارها في الشجرة، فمن المفيد أن نعرف من الجزء (أ) أن  $z$  لونها أحمر. سنستخدم الجزء (ب) لإظهار وجود العقدة  $z.p.p$  عندما نشير إليها في الأسطر 2 و 3 و 7 و 8 و 13 و 14.



**الشكل 4.13** عملية RB-INSERT-FIXUP. (أ) عقدة  $z$  بعد الإدراج. لما كانت  $z$  حمراء وأبوها  $z.p$  كذلك، فإنه يحدث خرق للخاصية 4. ولما كان عم  $z$  وهو  $y$  أحمر، فإن الحالة الأولى من الرماز case 1 تنطبق. نعيد تلوين العقدة وننقل المؤشر  $z$  إلى الأعلى ضمن الشجرة، فتنتج الشجرة الظاهرة في (ب). مرة أخرى،  $z$  وأبوها كلاهما حمراوان، لكن عم  $z$  وهو  $y$  أسود. ولما كانت  $z$  هي الابن الأيمن لـ  $z.p$ ، فإن الحالة الثانية case 2 تنطبق. نجري دورانا إلى اليسار، فنظهر الشجرة الناتجة في الشكل (ت). أصبح الآن  $z$  هو الابن الأيسر لأبيه، فتطبق الحالة الثالثة case 3. بإعادة التلوين وتطبيق دوران إلى اليمين تنتج الشجرة في (ث)، وهي شجرة حمراء-سوداء صحيحة.

تذكّر أننا نحتاج إلى إظهار أنَّ لامتغير الحلقة صحيح قبل التكرار الأول للحلقة، وأن كل تكرار يحافظ على لامتغير الحلقة، وأن لامتغير الحلقة يعطينا خاصية مفيدة في نهاية الحلقة.

نبدأ بمناقشة الاستبعاد والانهاء. ثم، وفي سياق دراستنا لكيفية عمل الحلقة بتفصيل أكبر، سنبرهن أن الحلقة تحافظ على اللامتغير في كل تكرار. سنبرهن أيضاً أن كل تكرار من الحلقة له نتيحتان محتملتان: إما أن يتنقل المؤشر  $z$  إلى الأعلى في الشجرة، وإما أن يجري بعض الدورانات ثم تنتهي الحلقة.

الاستبعاد: قبل التكرار الأول للحلقة، بدأنا بشجرة حمراء-سوداء بدون عروقات، وأضفنا عقدة حمراء  $z$ . نبيّن أن كل جزء من اللامتغير محقق عند استدعاء الإجراء RB-INSERT-FIXUP:

أ. عند استدعاء الإجراء RB-INSERT-FIXUP، فإن العقدة  $z$  هي العقدة الحمراء التي أضيفت.

ب. إذا كان  $z.p$  هو الجذر، فإنه يكون في البداية أسود ولا يتغير لونه قبل استدعاء RB-INSERT-FIXUP.

ت. رأينا سابقاً أنَّ الخصائص 1 و 3 و 5 محققة عند استدعاء RB-INSERT-FIXUP.

إذا عرفت الشجرة الخاصة 2، فإنَّ الجذر الأحمر يجب أن يكون هو العقدة  $z$  المضافة حديثاً، وهي العقدة الداخلية الوحيدة في الشجرة. ولما كان لكلٍّ من أبي  $z$  وابنيها قيمة الحارس، الذي هو أسود، فإنَّ الشجرة لا تُخرق الخاصية 4 أيضاً. ومن ثَمَّ فإنَّ هذا الخرق للخاصية 2 هو الخرق الوحيد للخصائص الحمراء-السوداء في كامل الشجرة.

إذا عرفت الشجرة الخاصة 4، فإنه لما كان ابنا العقدة  $z$  هما حارسين أسودين وليس في الشجرة عروقات أخرى قبل إضافة  $z$ ، فإنَّ الخرق يجب أن يحصل لأن  $z$  و  $z.p$  كليهما أحمران. وفيما عدا ذلك لا تُخرق الشجرة خصائص حمراء-سوداء أخرى.

الانهاء: عندما تنتهي الحلقة فإنَّ ذلك يكون بسبب أن  $z.p$  أسود اللون. (إذا كان  $z$  هو الجذر فإنَّ  $z.p$  هو الحارس  $T.nil$ ، وهو أسود.) لذلك لا تُخرق الشجرة الخاصة 4 عند انتهاء الحلقة. والخاصية الوحيدة التي يمكن أن تُخرق نغفها بوجود لامتغير الحلقة هي الخاصية 2. يستعيد السطر 16 هذه الخاصية أيضاً بحيث تكون جميع الخصائص الحمراء-السوداء محققة عند انتهاء RB-INSERT-FIXUP.

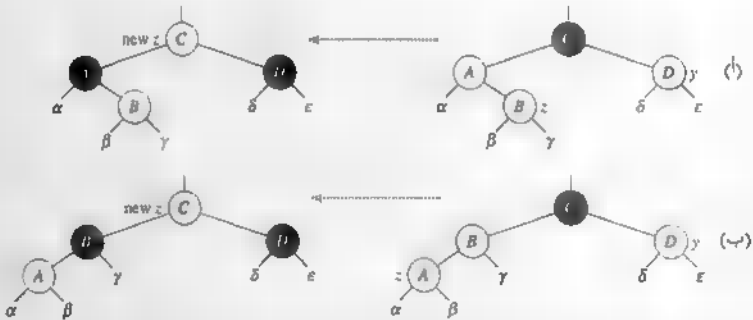
المحافظة: نحتاج فعلًا لاعتبار ست حالات في حلقة **while**، غير أنَّ ثلاثاً منها مناظرة للثلاث الأخرى، بحسب ما يتحدّد في السطر 2 من كون أبو  $z$  وهو  $z.p$  ابناً أيسر أو ابناً أيمن لجد  $z.p.p$ . وقد أعطينا فقط الرمز الخاص بالحالة التي يكون فيها  $z.p$  ابناً أيسر. إن العقدة  $z.p.p$  موجودة لأنه بحسب الجزء (ب) من لامتغير الحلقة، إذا كان  $z.p$  هو الجذر فإنه يكون أسود. ولما كنا ندخل إلى تكرار الحلقة فقط إذا كانت  $z.p$  حمراء، فإننا نعلم أنَّ  $z.p$  لا يمكن أن تكون هي الجذر، ومن ثَمَّ فإنَّ  $z.p.p$  موجودة.

تغير الحالة الأولى case 1 عن الحالتين 2 و 3 بلون ذرية الأب العائد لـ  $z$  أو "العم". إن السطر 3 يجعل العقدة  $y$  تشير إلى عم  $z$  وهو  $z.p.p.right$ ، في حين يختار السطر 4 لون  $y$ . فإذا كانت  $y$  حمراء تُنفذ الحالة 1، وإلا ينتقل التحكم إلى الحالتين 2 و 3. في جميع الحالات الثلاث، يكون جد  $z$  وهو  $z.p.p$  أسود، لأن أباه  $z.p$  لونه أحر، والخاصية 4 تُحترق بين  $z$  و  $z.p$  فقط.

### الحالة الأولى: عم $z$ (وهو $y$ ) أحمر

يُظهر الشكل 5.13 وضع الحالة 1 (الأسطر 8-5) التي تُنفذ عندما يكون كلٌّ من  $z.p$  و  $y$  حمرًا. ولما كان  $z.p.p$  أسود، فيمكننا تلوين كلٌّ من  $z.p$  و  $y$  بالأسود، وبذلك نحل مشكلة كون  $z$  و  $z.p.p$  حمرًا، ويمكننا تلوين  $z.p.p$  بالأحمر، وبذلك نحفظ الخاصية 5. ثم نكرر الحلقة `while` باعتبار  $z.p.p$  هو العقدة الجديدة  $z$ . ينتقل للوشر ■ مستويين إلى أعلى الشجرة. نبين الآن أن الحالة الأولى تُحافظ على لامتغير الحلقة في بداية التكرار التالي. نستخدم  $z$  للتعبير عن العقدة  $z$  في التكرار الحالي، ونستخدم  $z' = z.p.p$  للتعبير عن العقدة التي مسمى  $z$  في اختبار السطر 1 في التكرار التالي.

أ. لما كان هذا التكرار يلوّن  $z.p.p$  بالأحمر، فإن  $z'$  تكون حمراء في بداية التكرار التالي.



الشكل 5.13 الحالة الأولى من الإجراء RB-INSERT-FIXUP. تُحترق الخاصية 4 لأن  $z$  وأبائها  $z.p$  حمرًا. يُنفذ العمل نفسه سواء أكان (أ) أبناً لـ  $z$  أو (ب) أبناً لـ  $z.p.p$ . لكلٍّ من الأشجار الفرعية  $\alpha$  و  $\beta$  و  $\gamma$  و  $\delta$  و  $\epsilon$  جذر أسود، ولكل منها الارتفاع الأسود نفسه. يغيّر الرمز في الحالة الأولى ألوان بعض العقد، معاًفظاً على الخاصية 5: جميع المسارات البسيطة النازلة من عقدة إلى ورقة تحتوي العدد نفسه من العقد السوداء. تتابع حلقة `while` مع العقدة الجدل  $z$  وهي  $z.p.p$  باعتبارها عقدة ■ جديدة. يمكن أن يحدث الآن أي عرق للخاصية 4 فقط بين العقدة الجديدة  $z$  (الحمراء) وأبيها، إذا كان أحر أيضاً.

ب. العقدة  $p.z'$  هي  $p.p.p.z$  في هذا التكرار، ولونها لا يتغير. فإذا كانت هي الجذر فلوئها كان أسود قبل هذا التكرار، ويبقى كذلك في بداية التكرار الذي يليه.

ت. سبق أن بينّا أن الحالة الأولى تحافظ على الخاصية 5، ولا تحدث عرقاً للخاصتين 1 أو 3.

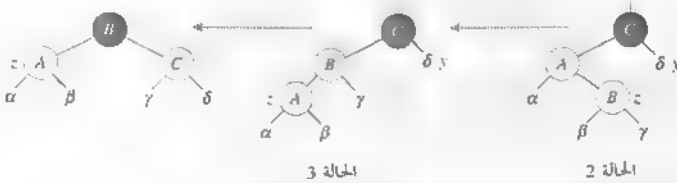
إذا كان  $z'$  هو الجذر في بداية التكرار التالي، فتكون الحالة الأولى قد صحتت الحرق الوحيد للخاصية 4 في هذا التكرار. ولأن  $z'$  لونها أحمر وهي الجذر، فإن الخاصية 2 تصبح هي الخاصية الوحيدة المخروقة ويعود سبب عرقها إلى  $z'$ .

إذا لم يكن  $z'$  هو الجذر في بداية التكرار التالي، فإن الحالة الأولى لا تكون قد تسببت بحرق الخاصية 2. صحتت الحالة الأولى الحرق الوحيد للخاصية 4 الذي كان موجوداً في بداية هذا التكرار. ثم جعلت  $z'$  حمراء وتركزت  $p.z'$  كما هي. فإذا كانت  $z'.p$  سوداء فلا حرق في الخاصية 4. أما إذا كانت  $z'.p$  حمراء فإن تلوين  $z'$  بالأحمر يتسبب بحرق واحد في الخاصية 4 بين  $z'$  و  $z'.p$ .

**الحالة الثانية:  $z$  هو ابن أيمن وعُقه  $y$  أسود**

**الحالة الثالثة:  $z$  هو ابن أيسر وعُقه  $y$  أسود**

في الحالتين الثانية والثالثة عم  $z$  هو  $y$  ولونه أسود. نُفرّق بين الحالتين بحسب كون  $z$  ابناً أيمن أو أيسر لـ  $p.z$ . تولّف الأسطر 10-11 الحالة الثانية، التي تظهر في الشكل 6.13 مع الحالة الثالثة. في الحالة الثانية، العقدة 2 هي ابن أيمن لأبيه. نستخدم فوراً دوراناً إلى اليسار لتحويل الوضع إلى الحالة الثالثة (الأسطر 12-14)، التي تكون فيها العقدة  $z$  ابناً أيسر. ولما كان كل من  $z$  و  $p.z$  حمراوان، فإن الدوران لا يؤثر على الارتفاع الأسود للعقد ولا على الخاصية 5. وسواء دخلنا الحالة الثالثة مباشرة أو عبر الحالة الثانية، فإن عم  $z$  يكون أسود،



**الشكل 6.13** الحالتان الثانية والثالثة للإجراء RB-INSERT-FIXUP. كما في الحالة الأولى، يُحرق الخاصية 4 إما في الحالة الثانية وإما في الثالثة لأن  $z$  وأبناها  $p.z$  حمراوان. لكل من الأشجار الفرعية  $\alpha$ ,  $\beta$ ,  $\gamma$  و  $\delta$  جذر أسود ( $\alpha$ ) و  $\beta$  و  $\gamma$  من الخاصية 4، و  $\delta$  لأننا بغير ذلك نكون في الحالة الأولى، ولكل منها الارتفاع الأسود نفسه. تتحول الحالة الثانية إلى الحالة الثالثة بدوراناً إلى اليسار يحافظ على الخاصية 5: جميع المسارات النازلة من عقدة إلى ورقة لها العدد نفسه من العقد السوداء. تسبّب الحالة الثالثة بعض التغيير في الألوان ودوراناً إلى اليمين يحافظ أيضاً على الخاصية 5. ثم تنتهي حلقة **while** لأن الخاصية 4 محققة: لم يعد ثمة عقدتان حمراوان في سطر.

لأننا نغير ذلك نكون قد نفذنا الحالة الأولى. إضافة إلى ذلك، فإن العقدة  $z.p.p$  موجودة، لأننا نبيّن أن هذه العقدة كانت موجودة عند تنفيذ السطرين 2 و 3، وبعد نقل  $z$  إلى المستوى الأعلى في السطر 10 ثم نقله إلى المستوى الأدنى في السطر 11، لم تتغير هوية  $z.p.p$ . وفي الحالة الثالثة، ننقذ بعض التغيير في الألوان ودوراناً إلى اليمين يحافظ على الخاصية 5، وبذلك نكون قد انتهينا، لأنه لم يعد لدينا عقدتان حمراوان في أي سطر. هذا وإن الحلقة **while** لا تتكرر مرة أخرى لأن  $z.p$  أصبحت سوداء الآن.

نبيّن الآن أن الحالتين الثانية والثالثة تحافظان على الاستمرارية للحلقة. (كما نبيّن للتو، فإن  $z.p$  ستكون سوداء بعد الاختيار التالي في السطر 1، ولن يُنقذ جسم الحلقة ثانية.)

أ. نجعل الحالة الثانية  $z$  تؤثر على  $z.p$  ذات اللون الأحمر. لا تجري تعديلات أخرى على  $z$  أو على لونه في الحالتين الثانية والثالثة.

ب. نجعل الحالة الثالثة العقدة  $z.p.p$  سوداء، فإذا كانت هي الجذر في بداية التكرار التالي، كان لونها أسود.

ت. كما في الحالة الأولى، نحافظ الحالتان الثانية والثالثة على الخصائص 1 و 3 و 5.

ولمّا لم تكن العقدة  $z$  هي الجذر في الحالتين الثانية والثالثة، فإننا نعلم أنه ليس هناك حرق للخاصية 2. فالحالتين الثانية والثالثة لا تسببان حرقاً للخاصية 2، لأن العقدة الوحيدة التي أصبحت حمراء تصبح ابناً لعقدة سوداء نتيجة للدوران في الحالة الثالثة.

نُصحّح الحالتان الثانية والثالثة الحرق الوحيد للخاصية 4، ولا تُدخلان حرقاً آخر.

بعد أن أظهرنا أن كل تكرار للحلقة يحافظ على الاستمرارية، نكون قد برهنّا أنّ الإجراء RB-INSERT-FIXUP يستعيد الخصائص الحمراء-السوداء استعادةً صحيحة.

### التحليل

ما هو زمن تنفيذ الإجراء RB-INSERT؟ لما كان ارتفاع الشجرة الحمراء-السوداء المولفة من  $n$  عقدة هو  $O(\lg n)$ ، فإنّ الأسطر 1-16 من RB-INSERT تستغرق زمناً  $O(\lg n)$ . في الإجراء RB-INSERT-FIXUP، تتكرر حلقة **while** فقط إذا نُقّدت الحالة الأولى، ثم ينتقل المؤشر  $z$  مستويين إلى الأعلى في الشجرة. لذلك فإنّ العدد الكلي لمرات تنفيذ الحلقة **while** يمكن أن يكون  $O(\lg n)$ . وهكذا فإنّ RB-INSERT يستغرق زمناً إجماليّاً  $O(\lg n)$ . وهو مع ذلك لا يمكن أن يقوم بأكثر من دورتين لأن حلقة **while** تنتهي إذا نُقّدت الحالة الثانية أو الثالثة.

### تمارين

#### 1-3.13

في السطر 16 من الإجراء RB-INSERT، نحدّد لون العقدة للدرجة حديثاً بالأحمر. لاحظ أننا لو اخترنا



وضعها بالأسود، فإنَّ الخاصية 4 من خصائص الشجرة الحمراء-السوداء لن تُحقَّق. لماذا لم تُحقَّر تلوين  $z$  بالأسود؟

### 3-3.13

أظهر الأشجار الحمراء-السوداء الناتجة عن إدراج متتابع للمفاتيح 41 و 38 و 31 و 12 و 19 و 8 في شجرة حمراء-سوداء فارغة في البداية.

### 3-3.13

افترض أنَّ الارتفاع الأسود لكلٍّ من الأشجار الفرعية  $\blacksquare$  و  $\beta$  و  $\gamma$  و  $\delta$  و  $\epsilon$  في الشكلين 5.13 و 6.13 هو  $k$ . علِّم كل عقدة في الشكلين بارتفاعها الأسود للتحقُّق من أن التحويل المشار إليه يحافظ على الخاصية 5.

### 4-3.13

يشعر أحد المدرسين بالقلق من إمكان وضع اللون RED في  $T.nil.color$  في الإجراء RB-INSERT-FIXUP. في هذه الحالة لا يسبب الاختيار في السطر 1 إنهاء الحلقة عندما تكون  $z$  هي الجذر. يبيِّن أنَّ قلق المدرس لا أساس له بإثبات أنَّ RB-INSERT-FIXUP لا يضع RED في  $T.nil.color$  أبداً.

### 5-3.13

لنأخذ شجرة حمراء-سوداء تتكوَّن من إدراج  $n$  عقدة باستخدام RB-INSERT. بيِّن أنَّه إذا كان  $n > 1$  كانت في الشجرة عقدة واحدة حمراء على الأقل.

### 6-3.13

اقترح طريقة لتفجير RB-INSERT بفعالية إذا لم يتضمن غنيل الأشجار الحمراء-السوداء تخزين مؤشرات إلى الأب.

## 4.13 الحذف

كما في العمليات الأساسية الأخرى على شجرة حمراء-سوداء ذات  $n$  عقدة، يستغرق الحذف زمنًا  $O(\lg n)$ ، إلا أنَّ حذف عقدة من شجرة حمراء-سوداء أكثر تعقيداً يقليل من إدراج عقدة.

يعتمد إجراء حذف عقدة من شجرة حمراء-سوداء على الإجراء TREE-DELETE (المقطع 3.12). نحتاج أولاً إلى مواءمة الإجراء الفرعي TRANSPLANT الذي يستدعيه TREE-DELETE بحيث يُطبَّق على شجرة حمراء-سوداء:

RB-TRANSPLANT( $T, u, v$ )

```

1  if  $u.p == T.nil$ 
2       $T.root = v$ 
3  elseif  $u == u.p.left$ 
```

```

4      u.p.left = v
5  else u.p.right = v
6      v.p = u.p

```

يختلف الإجراء RB-TRANSPLANT عن TRANSPLANT من ناحيتين. الأولى أنَّ السطر 1 يستخدم الحارس  $T.nil$  بدلاً من  $nil$ ، والثانية أن الإسناد إلى  $v.p$  يحدث في السطر 3 دون شرط: يمكننا الإسناد إلى  $v.p$  وإن كان  $v$  يشير إلى الحارس. في الحقيقة سنستفيد من إمكان الإسناد إلى  $v.p$  عندما يكون  $v = T.nil$ . إن الإجراء RB-DELETE يشبه الإجراء TREE-DELETE، ولكن بزيادة أسطر من شبه الرماز. تفتي بعض الأسطر الإضافية أثر عقدة  $y$  قد تسبب في عرق الخصائص الحمراء-السوداء. وعندما نريد حذف عقدة  $z$  التي لها أقل من ابنين، فإننا نحذف من الشجرة ونريد أن تصبح  $y$  هي  $z$ . أما عندما يكون ل  $z$  ابنان، فيجب عندها أن تكون  $y$  هي العقدة التالية ل  $z$ ، وأن تحتل  $y$  موضع  $z$  في الشجرة. كذلك نسجل لون  $y$  قبل حذفها من الشجرة أو انتقالها ضمنها، ونتبع العقدة  $x$  التي تنتقل إلى موضع  $y$  الأصلي في الشجرة، لأن العقدة  $x$  قد تسبب أيضًا عروقًا في الخصائص الحمراء-السوداء. بعد حذف العقدة  $z$ ، يستدعي الإجراء RB-DELETE إجراء مساعدًا RB-DELETE-FIXUP يغير الألوان ويقوم بدوراتنا لاستعادة الخصائص الحمراء-السوداء.

RB-DELETE( $T, z$ )

```

1  y = z
2  y-original-color = y.color
3  if z.left == T.nil
4      x = z.right
5      RB-TRANSPLANT(T, z, z.right)
6  elseif z.right == T.nil
7      x = z.left
8      RB-TRANSPLANT(T, z, z.left)
9  else y = TREE-MINIMUM(z.right)
10     y-original-color = y.color
11     x = y.right
12     if y.p == z
13         x.p = y
14     else RB-TRANSPLANT(T, y, y.right)
15         y.right = z.right
16         y.right.p = y
17     RB-TRANSPLANT(T, z, y)
18     y.left = z.left
19     y.left.p = y
20     y.color = z.color
21 if y-original-color == BLACK
22     RB-DELETE-FIXUP(T, x)

```

ومع أنَّ RB-DELETE يتضمن عددًا من أسطر شبه الرماز يقارب ضعف عدد أسطر TREE-DELETE، فإنَّ الإجراءين لهما البنية الأساسية نفسها. يمكنك إيجاد كل سطر من TREE-DELETE ضمن RB-DELETE (بتغيير NIL إلى  $T.nil$  والاستعاضة عن كل استدعاءات TRANSPLANT باستدعاءات RB-TRANSPLANT)، متفادًا ضمن الشروط نفسها. وفيما يلي بقية الاختلافات بين الإجراءين:

- نحافظ على العقدة  $y$  على أنها العقدة التي يجري حذفها من الشجرة أو نقلها ضمنها. نجعل السطر 1 للعقدة  $y$  تشير إلى العقدة  $z$  عندما يكون  $z$  أقل من ابنين، ومن ثم فهي تُحذف. أما عندما يكون  $z$  ابنًا، فإننا نجد في السطر 9 أن  $y$  تشير إلى العقدة التي تلي  $z$  كما في TREE-DELETE تمامًا، وستحتل  $y$  موضع  $z$  في الشجرة.
- لما كان من الممكن أن يتغيّر لون  $y$ ، فإنَّ التحول  $y \leftarrow y\text{-original-color}$  يخرن لونها قبل حدوث أي تغيير. يقوم السطران 2 و 10 بتحديد قيمة لهذا التحول مباشرةً بعد الإسناد إلى  $y$ . عندما يكون  $z$  ابنًا، فإن  $z \neq y$ ، وتنقل العقدة  $y$  إلى للموضع الأصلي للعقدة ■ في الشجرة الحمراء-السوداء؛ يعطي السطر 20 للعقدة  $y$  لون  $z$  نفسه. ونحتاج إلى حفظ لون  $y$  الأصلي لاختباره في خطوة RB-DELETE؛ فإذا كان أسود فإنَّ حذف  $y$  أو نقلها قد يسبب عروقات في الخصائص الحمراء-السوداء.
- كما ذكرنا آنفًا، تتبع العقدة  $x$  التي تنتقل إلى موضع  $y$  الأصلي. أما الإسنادات في الأسطر 4 و 7 و 11 فتجعل  $x$  تشير إلى ابن  $y$  الوحيد أو إلى الحارس  $T.nil$  إذا لم يكن  $z$  أولاد (نذكر من المقطع 3.12 أنَّ  $y$  ليس لها ابن أبسر).
- لما كانت العقدة  $x$  تنتقل إلى موضع العقدة  $y$  الأصلي، فإنَّ الوصفة  $x.p \leftarrow x$  مهيأة دومًا لشير إلى الموضع الأصلي لأي  $y$  في الشجرة، وإن كانت العقدة  $x$  هي في الحقيقة الحارس  $T.nil$ . وفيما عدا الحالة التي يكون فيها  $z$  هو الأب الأصلي للعقدة  $y$  (وهذا يحدث فقط عندما يكون للعقدة  $z$  ابنان وتكون العقدة التالية لها  $y$  هي ابنها الأيمن)، يحدث الإسناد إلى  $x.p$  في السطر 6 من RB-TRANSPLANT. (لاحظ أنَّه عند استدعاء RB-TRANSPLANT في الأسطر 5 أو 8 أو 14 فإنَّ المتوسط الثالث الذي يجري تمريره بحال  $x$ .)
- مع ذلك، عندما يكون الأب الأصلي للعقدة  $y$  هو  $z$ ، فإننا لا نريد أن يشير  $x.p$  إلى الأب الأصلي للعقدة  $y$ ، لأننا بصدق حذف تلك العقدة من الشجرة. ولما كانت العقدة  $y$  تستقل إلى الأعلى لتأخذ موضع  $z$  في الشجرة، فإنَّ وضع  $y$  في  $x.p$  في السطر 13 سيجعل  $x.p$  يشير إلى الموضع الأصلي لأي العقدة  $y$  وإن كان  $x = T.nil$ .
- أخيرًا، إذا كانت العقدة  $y$  سوداء، فقد نكون أحدثنا عرقًا أو أكثر في الخصائص الحمراء-السوداء،

ولذلك نستدعي RB-DELETE-FIXUP في السطر 22 لاستعادة الخصائص الحمراء-السوداء. فإذا كانت  $y$  حمراء، بقيت الخصائص الحمراء-السوداء محققة عند حذف  $y$  أو نقلها، وذلك للأبواب التالية:

1. لم تتغير أية ارتفاعات سوداء في الشجرة.
  2. لم يجر وضع عقد حمراء متحلولة. ولما كانت  $y$  تأخذ مكان  $z$  في الشجرة وكذلك لون  $z$ ، فلا يمكن أن يصبح لدينا عقدتان حمراوان متجاورتان في موضع  $y$  الجديد في الشجرة. إضافة إلى ذلك، إذا لم تكن  $y$  الابن الأيمن للعقدة  $z$ ، فإن الابن الأيمن الأصلي لها  $x$  سيحل محلها في الشجرة. فإذا كانت  $y$  حمراء، لزم أن يكون  $x$  أسود، وبذلك لا يمكن أن يسبب تبديل  $x$  بـ  $y$  في أن تصبح عقدتان حمراوان متجاورتين.
  3. لما لم يكن بإمكان  $y$  أن تكون جذراً إذا كانت حمراء، فإن الجذر يبقى أسود.
- إذا كانت العقدة  $y$  سوداء، فيمكن أن تظهر ثلاث مشكلات تُحلّ باستدعاء RB-DELETE-FIXUP. أولاً، إذا كانت  $y$  هي الجذر وأصبح ابنه الأحمر لها هو الجذر الجديد، فنكون قد خرقنا الخاصية 2. ثانياً، إذا كان كلٌّ من  $x$  و  $p$  حمراوين، فنكون قد خرقنا الخاصية 4. ثالثاً، إن نقل  $y$  ضمن الشجرة بسبب نقص عقدة سوداء من كل مسار بسيط كان يتضمن  $y$ . وبذلك تُخرق الخاصية 5 من كل أسلاف  $y$  في الشجرة. نصحح خرق الخاصية 5 بالقول إن العقدة  $x$  التي تشغل الآن الموضع الأصلي للعقدة  $y$  لديها عقدة سوداء "إضافية". أي إننا إذا أضفنا 1 إلى عدد العقد السوداء في أي مسار بسيط يتضمن  $x$ ، فإن الخاصية 5 تتحقق بحسب هذا التفسير. وعندما نحذف أو نقل العقدة السوداء  $y$ ، "ندفع" بواسطتها في العقدة  $x$ . فللمشكلة الآن هي أن العقدة  $x$  لم تعد حمراء ولا سوداء، وذلك يخرق الخاصية 1. وبدلاً من ذلك، إما أن تكون العقدة  $x$  "سوداء مضاعفة" وإما "حمراء وسوداء" وتسهم في عدد العقد السوداء في المسارات البسيطة التي تحتوي  $x$  بمقدار 2 أو 1 بالترتيب. وسيتبقى واصف اللون  $color$  في العقدة  $x$  إما RED (إذا كانت  $x$  حمراء وسوداء) وإما BLACK (إذا كانت  $x$  سوداء مضاعفة). ونعتبر آخر فإن الأسود الإضافي في عقدة ما يؤثر في تأشير  $x$  عليها، وليس في واصف اللون  $color$ .

يمكننا الآن متابعة الإجراء RB-INSERT-FIXUP ونفحص كيفية إعادته الخصائص الحمراء-السوداء إلى شجرة البحث.

RB-DELETE-FIXUP( $T, x$ )

```

1  while  $x \neq T.root$  and  $x.color == BLACK$ 
2      if  $x == x.p.left$ 
3           $w = x.p.right$ 
4          if  $w.color == RED$ 
5               $w.color = BLACK$                                 // Case 1
6               $x.p.color = RED$                                 // Case 1

```

```

7      LEFT-ROTATE( $T, x.p$ )                                // Case 1
8       $\omega = x.p.right$                                        // Case 1
9      if  $\omega.left.color == BLACK$  and  $\omega.right.color == BLACK$ 
10      $\omega.color = RED$                                      // Case 2
11      $x = x.p$                                               // Case 2
12     else if  $\omega.right.color == BLACK$ 
13      $\omega.left.color = BLACK$                              // Case 3
14      $\omega.color = RED$                                      // Case 3
15     RIGHT-ROTATE( $T, \omega$ )                               // Case 3
16      $\omega = x.p.right$                                      // Case 3
17      $\omega.color = x.p.color$                                // Case 4
18      $x.p.color = BLACK$                                      // Case 4
19      $\omega.right.color = BLACK$                              // Case 4
20     LEFT-ROTATE( $T, x.p$ )                                   // Case 4
21      $x = T.root$                                           // Case 4
22     else (same as then clause with "right" and "left" exchanged)
23      $x.color = BLACK$ 

```

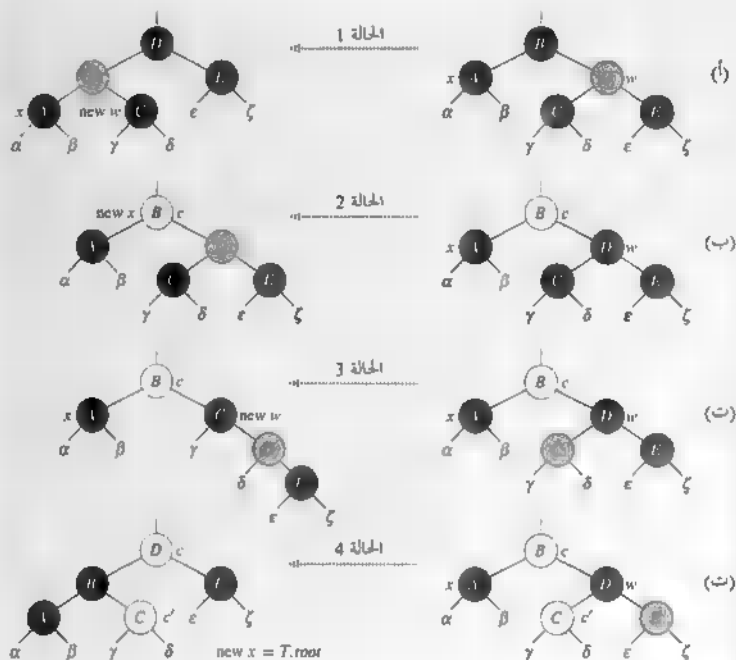
يستعيد الإجراء RB-DELETE-FIXUP الخواص 1 و 2 و 4. يُطلب إليك في التمرينين 14.13 و 14.13 أن تثبت أن الإجراء يستعيد الخاصيتين 2 و 4، لذلك فإننا سنركز في ما تبقى من هذا المقطع على الخاصية 1. ويمثل الغرض من حلقة **while** في الأسطر 1-22 في نقل السواد الإضافي إلى أعلى الشجرة إلى أن:

1. يشير  $x$  إلى عقدة حمراء-سوداء، وفي هذه الحالة نلزم  $x$  فقط بالأسود في السطر 23.
2. يشير  $x$  إلى الجذر، وفي هذه الحالة "نحذف" السواد الإضافي فقط.
3. نخرج من الحلقة بعد إجراء دورانات وعمليات إعادة تلوين مناسبة.

وضمن الحلقة **while**، تشير  $x$  دومًا إلى عقدة سوداء مضاعفة لا تنتمي إلى الجذر. نُحدد في السطر 2 إذا كان  $x$  هو أبنا أيسر أو أيمن لأبيه  $x.p$ . (أعطينا الرمز للحالة التي يكون فيها  $x$  أبنا أيسر؛ أما الحالة التي يكون فيها  $x$  أبنا أيمن —السطر 22— فهي حالة منظرية.) نحافظ على مؤشر  $w$  على العقدة المجاورة لـ  $x$ . لما كانت  $x$  ذات سواد مضاعف، فإن العقدة  $w$  لا يمكن أن تكون  $T.nil$ ؛ وإلا فإن عدد العقد السوداء على المسار البسيط من  $x.p$  إلى الورقة  $w$  (ذات السواد الوحيد) سيكون أصغر من عددها في المسار البسيط من  $x.p$  إلى  $x$ .

تظهر الحالات الأربع<sup>2</sup> للموجودة في الرمز في الشكل 7.13. وقبل دراسة كل حالة بالتفصيل، لنلق نظرة عامة على كيفية التحقق من حفاظ التحويل في كل حالة على الخاصية 5. الفكرة الرئيسية هي أن التحويل المطبق يحافظ على عدد العقد السوداء (ومنها السواد الإضافي الخاص بـ  $x$ ) في كل حالة انطلاقًا من جذر

<sup>2</sup> كما في RB-INSERT-FIXUP، ليست الحالات في RB-DELETE-FIXUP حالات إقصاء متبادل.



**الشكل 7.13 حالات الحلقة while في الإجراء RB-DELETE-FIXUP.** العقد المظلمة تكون واصفة اللون *color* فيها BLACK، أما العقد المظلمة تظليلاً كيميائياً فواصفة اللون *color* فيها RED، أما العقد المظلمة تظليلاً فانتحاء فواصفة اللون فيها ممثلة بـ  $c$  أو  $c'$ ، الذي يمكن أن يكون RED أو BLACK. تمثل الأحرف  $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$  أشجاراً فرعية اعتباطية. كل حالة تقوم بتحويل التشكيلة الموجودة إلى اليسار إلى التشكيلة الموجودة إلى اليمين، بتغيير بعض الألوان و/أو إجراء دوران. أي عقدة تشير إليها  $x$  يكون فيها سواءً إضافي، وتكون إما سوداء مضاعفة وإما حمراء-سوداء. الحالة الوحيدة التي تسبب تكرار الحلقة هي الحالة الثانية 2 case. (أ) يجري تحويل الحالة الأولى إلى الحالات 1 أو 3 أو 4 بتبديل ألوان العقدتين  $B$  و  $D$  وإجراء دوران إلى اليسار. (ب) في الحالة الثانية يجري نقل السوداء الإضافي للممثل بالخوشر  $x$  إلى أعلى الشجرة بتلوين العقدة  $D$  بالأحمر وجعل  $x$  تلوشر على العقدة  $B$ . وإذا دخلنا الحالة الثانية عبر الحالة الأولى، فإن الحلقة *while* تنتهي لأن العقدة الجديدة  $x$  هي حمراء-سوداء، ومن ثم فإن القيمة  $c$  لواصفة اللون فيها هي RED. (ت) تُحوّل الحالة الثالثة إلى الحالة الرابعة بتبديل ألوان العقدتين  $C$  و  $D$  وإجراء دوران يميني. (ث) تحذف الحالة الرابعة السوداء الإضافي للممثل بالخوشر  $x$  بتغيير بعض الألوان وإجراء دوران إلى اليسار (دون حرق الخصائص الحمراء-السوداء)، وتنتهي الحلقة.

الشجرة الفرعية المرسومة (وباعتباره متضمنًا) إلى كلاً من الأشجار الفرعية  $\alpha, \beta, \dots, \gamma$ . لذلك إذا كانت الخاصة 5 محققة قبل التحويل، فستظل محققة بعده. فمثلاً في الشكل 7.13 (أ) الذي يوضّح الحالة الأولى case 1، نلاحظ أنّ عدد العقد السوداء من الجذر إلى أي من الشجرتين الفرعيتين  $\alpha$  أو  $\beta$  هو 3، قبل التحويل وبعده. (تذكّر مرة أخرى أنّ العقدة  $x$  تضيف عقدة سوداء إضافية.) وبالمثل، فإنّ عدد العقد السوداء من الجذر إلى أي من  $\gamma$  و  $\delta$  و  $\varepsilon$  و  $\zeta$  هو 2، قبل التحويل وبعده. في الشكل 7.13 (ب)، يجب أن يأخذ العد بالحساب القيمة  $c$  لوصفة اللون  $color$  لجذر الشجرة الفرعية المرسومة، التي يمكن أن تكون RED أو BLACK. إذا عرّفنا  $count(RED) = 0$  و  $count(BLACK) = 1$ ، فإنّ عدد العقد السوداء من الجذر إلى  $\alpha$  هو  $2 + count(c)$ ، قبل التحويل وبعده. في هذه الحالة، تأخذ العقدة الجديدة  $x$  بعد التحويل القيمة  $c$  في واصفة اللون  $color$ ، لكن هذه العقدة هي في الحقيقة حمراء-سوداء (إذا كان  $c = RED$ ) أو سوداء مضاعفة (إذا كان  $c = BLACK$ ). يمكن التحقق من الحالات الأخرى بطريقة مشابهة (انظر التمرين 5-4.13).

#### الحالة الأولى: $w$ المجاورة لـ $x$ حمراء

نحدث الحالة الأولى (الأسطر 8-5 من RB-DELETE-FIXUP والشكل 7.13 (أ)) عندما تكون العقدة  $w$  المجاورة للعقدة  $x$  حمراء. وإذا إن  $w$  يجب أن يكون لها أبناء سوداء، فيسكننا تبديل لوني لـ  $w$  و  $x$ . ثم إجراء دوران إلى اليسار حول  $x$ .  $p$  دون خرق أي من الخصائص الحمراء-السوداء. أصبحت العقدة الجديدة المجاورة لـ  $x$ ، وهي أحد أبناء  $w$  قبل الدوران، الآن سوداء، ومن ثم تكون فد حوّلنا الحالة الأولى إلى الحالة 2 أو 3 أو 4.

نحدث الحالات 2 و 3 و 4 عندما تكون العقدة  $w$  سوداء؛ وتتمايز بالآتي أبناء  $w$ .

#### الحالة الثانية: $w$ المجاورة لـ $x$ سوداء، وابناها أسودان

في الحالة الثانية (الأسطر 10-11 من RB-DELETE-FIXUP والشكل 7.13 (ب)) يكون ابنا  $w$  أسودان. ولما كانت  $w$  أيضاً سوداء، فإنّخذ سوداءً واحداً من  $x$  ومن  $w$  لتصبح  $x$  بسواد واحد وتصبح  $w$  حمراء. للتعويض عن حذف سواد واحد من  $x$  ومن  $w$ ، نريد أن نضيف سواداً إضافياً إلى  $x$ .  $p$ ، الذي كان في الأصل أحمر أو أسود. نقوم بذلك بتكرار الحلقة **while** مع اعتبار  $x$ .  $p$  هي العقدة الجديدة  $x$ . لاحظ أننا إذا دخلنا الحالة الثانية عبر الحالة الأولى، فإنّ العقدة الجديدة  $x$  هي حمراء-سوداء. لأن العقدة الأصلية  $x$ .  $p$  كانت حمراء. إذن فالقيمة  $c$  لوصافة اللون  $color$  في العقدة الجديدة  $x$  هي RED، وتنتهي الحلقة عندما نعتبر شرط التكرار. ثم نلوّن العقدة الجديدة  $x$  بلون أسود (وحيث في السطر 23).

#### الحالة الثالثة: $w$ المجاورة لـ $x$ سوداء، وابنها الأيسر أحمر وابنها الأيمن أسود

نحدث الحالة الثالثة (الأسطر 13-16 والشكل 7.13 (ت)) عندما تكون  $w$  سوداء، وابنها الأيسر أحمر وابنها

الأيمن أسود. يمكننا تبديل لوني  $w$  وابنها الأيسر  $w.left$ . ثم إجراء دوران إلى اليمين حول  $w$  دون خرق أيٍّ من الخصائص الحمراء-السوداء. العقدة الجديدة  $w$  المجاورة لـ  $x$  هي الآن عقدة سوداء لها ابن أيمن أحمر، ومن ثمَّ نكون قد حولنا الحالة 3 إلى الحالة 4.

**الحالة الرابعة:**  $w$  المجاورة لـ  $x$  سوداء، وابنها الأيمن أحمر

تحدث الحالة الرابعة (الأسطر 17-21 والشكل 7.13(ث)) عندما تكون العقدة  $w$  المجاورة لـ  $x$  سوداء وابنها الأيمن أحمر. بإجراء بعض التغييرات في الألوان وتنفيذ دوران إلى اليسار حول  $x.p$ ، يمكننا حذف الأسود الإضافي في  $x$  لتصبح وحيدة السواد دون خرق أيٍّ من الخصائص الحمراء-السوداء. وبافتراض أن  $x$  هي الجذر تنتهي حلقة **while** عند اختبارها شرط التكرار.

### التحليل

ما هو زمن تنفيذ RB-DELETE؟ لِمَا كَانَ ارتفاع الشجرة الحمراء-السوداء للوظيفة من  $n$  عقدة هو  $O(\lg n)$ ، فإنَّ التكلفة الإجمالية للإجراء دون استدعاء RB-DELETE-FIXUP تستغرق زمنًا  $O(\lg n)$ . مع الإجراء RB-DELETE-FIXUP، تنتهي الحالات 1 و 3 و 4 بعد أداء عدد ثابت من تغييرات اللون وثلاثة دورانات على الأكثر. والحالة 2 هي الحالة الوحيدة التي يمكن أن تتكرر فيها الحلقة **while**، ومن ثمَّ ينتقل المؤشر  $x$  إلى أعلى الشجرة عددًا من المرات لا يتجاوز  $O(\lg n)$  مرة، وبدون دورانات. لذلك، فإنَّ الإجراء RB-DELETE-FIXUP يستغرق زمنًا  $O(\lg n)$ ، ويؤدي ثلاثة دورانات على الأكثر، ويكون الزمن الإجمالي للإجراء RB-DELETE إذن  $O(\lg n)$  أيضًا.

### تعاريف

#### 1-4.13

برهن أنه بعد تنفيذ RB-DELETE-FIXUP، يجب أن يكون جذر الشجرة أسود.

#### 2-4.13

برهن أنه إذا كان كل من  $x$  و  $x.p$  حمراوان في RB-DELETE، فإنَّ الخاصية 4 تُسترد باستدعاء RB-DELETE-FIXUP( $T, x$ ).

#### 3-4.13

أوجدت في التمرين 2-3.13 الشجرة الحمراء-السوداء التي تنتج عن إدراج متتابع للمفاتيح التالية 8، 19، 12، 31، 41، في شجرة فارغة في البداية. برِّ الآن الأشجار الحمراء-السوداء التي تنتج عن الحذف المتتابع للمفاتيح بحسب الترتيب 8، 19، 12، 31، 41.

#### 4-4.13

في أية أسطر من رماز RB-DELETE-FIXUP يمكننا تفحص أو تعديل الحارس  $T.nil$ ؟



## 5-4.13

في كل من حالات الشكل 7.13، حدد عدد العقد السوداء انطلاقاً من جذر الشجرة الفرعية المرسومة إلى كلٍ من الأشجار الفرعية  $\alpha, \beta, \dots, \gamma$ ، وتأكد أن عددها يظل نفسه بعد التحويل. وإذا كانت واصفة اللون *color* لعقدة ما هي  $c$  أو  $c'$ ، فاستخدم التدوين  $\text{count}(c)$  أو  $\text{count}(c')$  رمزياً في إجابتك.

## 6-4.13

يعتقد الأستاذان Skelton و Baron بأن لون العقدة  $x.p$  يمكن أن لا يكون أسود في بداية الحالة الأولى من RB-DELETE-FIXUP. فإذا كان الأستاذان محقّقين، تكون الأسطر 6-5 خاطئة. أثبت أن  $x.p$  يجب أن يكون أسود في بداية الحالة الأولى، بحيث يدرك الأستاذان أنه ليس ثمة ما يقلقنا بشأنه.

## 7-4.13

افترض أن عقدة  $x$  أدرجت في شجرة حمراء-سوداء باستخدام RB-INSERT، ثم حُذفت مباشرة باستخدام RB-DELETE. هل الشجرة الحمراء-السوداء الناتجة هي نفسها الشجرة الحمراء-السوداء الأولى؟ علّل جوابك.

## مسائل

## 1-13 مجموعات ديناميكية دائمة

في سياق تتبع خوارزمية ما، نجد أحياناً أننا نحتاج إلى الاحتفاظ بالنسخ السابقة لمجموعة ديناميكية dynamic set عند تعديلها. تُسمى هذه المجموعة دائمة *persistent*. تعتمد إحدى طرق تنجيز مجموعة دائمة على نسخ المجموعة كاملة كلما جرى تعديلها، لكن هذا النهج يمكن أن يُعطى البرنامج وأن يستهلك حجباً كبيراً. على أنه يمكننا أحياناً أن نفعل شيئاً أفضل بكثير من هذا النهج.

افترض  $S$  مجموعة دائمة مع العمليات INSERT, DELETE, SEARCH التي ندرجها باستخدام أشجار بحث ثنائية كما في الشكل 8.13(أ). نحفظ جذر منفصل لكل نسخة من المجموعة. ولإدراج المفتاح 5 في المجموعة، ننشئ عقدة جديدة مفتاحها 5. تصبح هذه العقدة الابن الأيسر لعقدة جديدة مفتاحها 7، لأننا لا نستطيع تغيير العقدة الموجودة سابقاً ذات المفتاح 7. وبالمثل، فإن العقدة الجديدة ذات المفتاح 7 تصبح الابن الأيسر لعقدة جديدة مفتاحها 8 وإنها الأيمن هو العقدة الموجودة ذات المفتاح 10. تصبح العقدة الجديدة ذات المفتاح 8 بدورها الابن الأيمن للجذر جديد  $r$  ذي المفتاح 4 وله ابن أيسر هو عقدة موجودة مفتاحها 3. وهكذا، فنحن ننسخ فقط جزءاً من الشجرة ونشترك ببعض العقد مع الشجرة الأصلية، كما يظهر في الشكل 8.13(ب).

لنفترض أن لكل عقدة في الشجرة الواصفات *key* و *left* و *right*، ولكن ليس لها أب. (انظر أيضاً التمرين 6-3.13)



**الشكل 8.13** (أ) شجرة بحث ثنائية مغابقتها 2، 3، 4، 7، 8، 10. (ب) شجرة البحث الثنائية الدائمة الناتجة عن إدراج المفتاح 5. تتألف أمدت نسخة من مجموعة من العقد التي يمكن الوصول إليها من الجذر ٣، وتتألف النسخة السابقة من العقد التي يمكن الوصول إليها من ٢. أضيفت العقد المظلة تظليلاً كلياً عند إدراج المفتاح 5.

أ. حدّد العقد الذي نحتاج إلى تغييرها لإدراج مفتاح  $k$ ، أو حذف عقدة  $y$  في شجرة بحث ثنائية دالة عامة.

ب. إذا كان لديك شجرة دائمة  $T$  ومفتاح  $k$  للإدراج، فاكتمل إجراء PERSISTENT-TREE-INSERT بعيد شجرة جديدة دائمة  $T'$  ناتجة عن إدراج  $k$  في  $T$ .

ت. إذا كان ارتفاع شجرة البحث الثنائية الدائمة  $T$  هو  $h$ ، فما هي متطلبات الزمن والحجم لتحيز `PERSISTENT-TREE-INSERT`؟ (تناسب متطلبات الحجم مع عدد العقد الجديدة المحصنة).

ث. افترض أننا ضمناً واصفة parent في كل عقدة. في هذه الحالة سيحتاج PERSISTENT-TREE-INSERT إلى القيام بعمليات نسخ إضافية. أثبت أن PERSISTENT-TREE-INSERT يستغرق زمناً وحجماً  $\Omega(n)$  حيث  $n$  هو عدد العقد في الشجرة.

ج. أظهر كيفية استخدام الأشجار الحمراء السوداء لضمان بقاء الزمن والحجم اللازمين للتنفيذ في أسوأ الحالات  $O(\lg n)$  لكل عملية إدراج أو حذف.

## 2-13 عملية الضم على الأشجار الحمراء-السوداء

نأخذ عملية الضم *join* مجموعتين ديناميكيتين  $S_1$  و  $S_2$  وعنصر  $x$  بحيث يكون لكل  $x_1 \in S_1$  و  $x_2 \in S_2$  لدينا  $x_1.key \leq x.key \leq x_2.key$ . تعيد هذه العملية مجموعة  $S = S_1 \cup \{x\} \cup S_2$ . نتحرى في هذه المسألة كيفية تنفيذ عملية الضم في الأشجار الحمراء-السوداء.

أ. إذا كان لدينا شجرة حمراء-سوداء  $T$ ، نخزن ارتفاعها الأسود باعتباره الوصف الجديد  $T.bh$ . بين أن  $RB-DELETE$  و  $RB-INSERT$  يمكنهما الاحتفاظ بهذا الوصف دون الحاجة إلى تخزين إضافي في عقد الشجرة، ودون زيادة زمن التنفيذ المقارب. برهن أنه يمكننا، في أثناء النزول عبر  $T$ ، تحديد الارتفاع الأسود لكل عقدة نزولها في زمن  $O(1)$  لكل عقدة جرت زيارتها.

نرغب بتنفيذ عملية  $RB-JOIN(T_1, x, T_2)$  التي تدثر  $T_1$  و  $T_2$  وتميد شجرة حمراء-سوداء  $T = T_1 \cup \{x\} \cup T_2$ . ليكن  $n$  العدد الكلي للعقد في  $T_1$  و  $T_2$ .

ب. افترض أن  $T_1.bh \geq T_2.bh$ . صف خوارزمية تعمل في زمن  $O(\lg n)$  بإمكانها أن تجد عقدة سوداء  $y$  في  $T_1$  يكون مفتاحها هو الأكبر بين العقد التي لها الارتفاع الأسود  $T_2.bh$ .

ت. لتكن  $T_y$  الشجرة الفرعية ذات الجذر  $y$ . صف كيف يمكن أن نحل  $T_y \cup \{x\} \cup T_2$  محل  $T_y$  في زمن  $O(1)$  دون تدمير خصائص شجرة البحث الثنائية.

ث. ما اللون الذي يمكن أن نلونه بـ  $x$  بحيث نحافظ على الخصائص الحمراء-السوداء 1 و 3 و 5؟ صف كيف يمكن فرض الخصائص 2 و 4 في زمن  $O(\lg n)$ .

ج. برهن أن الفرض في الجزء (ب) لا يضعف العمومية. صف الحالة المناظرة التي تظهر عندما يكون  $T_1.bh \leq T_2.bh$ .

ح. برهن أن زمن تنفيذ  $RB-JOIN$  هو  $O(\lg n)$ .

### 3-13 أشجار AVL

شجرة  $AVL$  هي شجرة بحث ثنائية متوازنة الارتفاع  $height\ balanced$ : ففي كل عقدة  $x$ ، تختلف ارتفاعات الأشجار الفرعية اليمنى واليسرى بمقدار 1 على الأكثر. ولتنحيز شجرة  $AVL$ ، نخفظ بوصف إضافي في كل عقدة  $x$ ،  $x.h$  هو ارتفاع العقدة  $x$ . ونفترض، كما في أية شجرة بحث ثنائية أخرى، أن  $T.root$  يشير إلى عقدة الجذر.

أ. برهن أن ارتفاع شجرة  $AVL$  ذات  $n$  عقدة هو  $O(\lg n)$ . (تلميح: برهن أنه في شجرة  $AVL$  ارتفاعها  $ch$  هناك على الأقل  $F_h$  عقدة، حيث  $F_h$  هو رقم فيبوناتشي  $Fibonacci$  من الرتبة  $h$ .)

ب. لإدراج عقدة ضمن شجرة  $AVL$ ، يجري أولاً وضع هذه العقدة في مكان مناسب ضمن ترتيب شجرة البحث الثنائية. بعد ذلك قد لا تبقى الشجرة متوازنة الارتفاع. وبالتحديد قد يختلف ارتفاعا الابن الأيسر والأيمن لبعض العقد بمقدار 2. صف إجراء  $BALANCE(x)$  يأخذ شجرة فرعية جذورها  $x$ ، وابناها الأيمن والأيسر متوازنا الارتفاع ويختلف ارتفاعهما بمقدار 2 على الأكثر، أي إن

ويعدّل الشجرة الفرعية ذات الجذر  $x$  لتصبح متوازنة الارتفاع.  
(تلميح: استخدم الدورانات.)

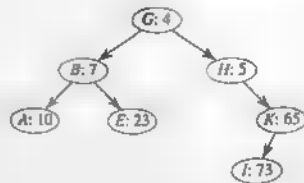
ت. باستخدام الجزء (ب)، صف إجراء عودياً  $AVL-INSERT(x, z)$  يأخذ عقدة  $x$  ضمن شجرة AVL وعقدة منشأة حديثاً  $z$  (سبق ملء مفتاحها)، ويضيف  $z$  إلى الشجرة الفرعية ذات الجذر  $x$ ، يحافظ على خاصية أن  $x$  هي جذر شجرة AVL. كما في الإجراء  $TREE-INSERT$  من المقطع 3.12، افترض أن  $z.key$  قد ملئ سابقاً، وأن  $z.left = NIL$  و  $z.right = NIL$ ؛ وافترض أيضاً أن  $z.h = \infty$ . إذن لإدراج العقدة  $z$  في شجرة AVL وهي  $T$ ، تستدعي  $AVL-INSERT(T.root, z)$ .

ث. بين أن تنفيذ عملية  $AVL-INSERT$  على شجرة AVL ذات  $n$  عقدة يستغرق زمناً  $O(\lg n)$ ، ويؤدي  $O(1)$  دورات.

#### 4-13 الأشجار المكوّمة

إذا أدرجنا مجموعة من  $n$  عنصراً في شجرة بحث ثنائية، فقد تكون الشجرة الناتجة شديدة الاختلال في التوازن، وهذا يؤدي إلى أزمنة بحث طويلة. ومع ذلك، وكما رأينا في المقطع 4.12 فإن أشجار البحث الثنائية للبيئة عشوائياً تميل إلى أن تكون متوازنة. لذلك يمكن اعتماد استراتيجية تبني - وسطياً - شجرة متوازنة من مجموعة ثابتة من العناصر تقوم على إجراء تبديل عشوائي للعناصر ثم إدراجها في الشجرة وفق ذلك الترتيب. ولكن ماذا يحدث لو لم تكن لدينا كل العناصر دفعة واحدة؟ إذا كنا نستقبل العناصر واحداً تلو الآخر، فهل يمكننا بناء شجرة بحث ثنائية عشوائياً بما؟

سندرس بنية معطيات تعطي جواباً إيجابياً عن هذا السؤال. الشجرة المكوّمة *treap* هي شجرة بحث ثنائية لها طريقة معدّلة في ترتيب العقد. يُظهر الشكل 9.13 مثلاً عنها. وكللتاد، تملك كل عقدة  $x$  في الشجرة مفتاحاً قيمته  $x.key$ . إضافة إلى ذلك، نسند الأفضلية  $x.priority$  وهي رقم عشوائي نختاره اختياراً مستقلاً لكل عقدة. نفترض أن كل الأفضليات متمايزة، وأن كل المفاتيح متمايزة أيضاً. تكون عقد الشجرة المكوّمة مرتبة بحيث تحقق للمفاتيح خصائص أشجار البحث الثنائية، وتحقق الأفضليات خاصية ترتيب الكومة وفق الأصغر:



الشكل 9.13 شجرة مكوّمة. كل عقدة  $x$  معلّمة بـ  $x.priority$  :  $x.key$ . فالجذر مثلاً مفتاحه  $G$  وأفضليته 4.

- إذا كان  $v$  أبناً لأيسر  $u$ ، فإن  $v.key < u.key$ .
- إذا كان  $v$  أبناً لأيمن  $u$ ، فإن  $v.key > u.key$ .
- إذا كان  $v$  أبناً لـ  $u$ ، فإن  $v.priority > u.priority$ .

سميت الشجرة بالشجرة المكثومة بسبب تركيب مجموعة الخصائص هذه؛ فهي تملك في آنٍ معا ميزات شجرة البحث الثنائية والكومة.)

من المفيد النظر إلى الأشجار المكثومة على النحو الآتي. افترض أننا نُدريج عقداً  $x_1, x_2, \dots, x_n$  مع مفاتيحها المرافقة ضمن شجرة مكثومة. فالشجرة المكثومة الناتجة هي الشجرة التي كانت ستشكّل فيما لو كانت العقد قد أُدرجت في شجرة بحث ثنائية عادية وفق ترتيب أفضليتها (المختارة عشوائياً). أي إن  $x_i.priority < x_j.priority$  تعني أننا أدرجنا  $x_i$  قبل  $x_j$ .

أ. لنكن لدينا مجموعة العقد  $x_1, x_2, \dots, x_n$  مع مفاتيحها المرافقة وأفضليتها، وجميعها متميزة. بيّن أن الشجرة المكثومة المرافقة لهذه العقد وحيدة.

ب. برهن أن الارتفاع المتوقع لشجرة مكثومة هو  $O(\lg n)$ ، وأن الزمن المتوقع للبحث عن قيمة في الشجرة المكثومة هو  $O(\lg n)$ .

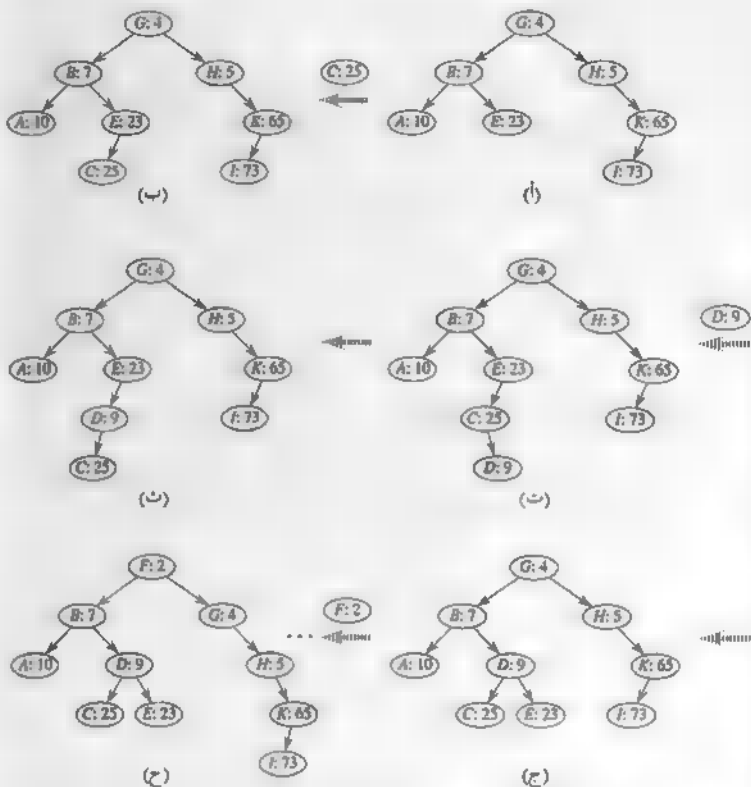
لتركيّف يمكن إدراج عقدة جديدة في شجرة مكثومة موجودة. نبدأ بإسناد أفضلية عشوائية للعقدة الجديدة. ثم نستخدمي خوارزمية الإدراج، التي نسميها TREAP-INSERT ونوضّح كيفية تشغيلها في الشكل 10.13.

ت. اشرح كيفية عمل TREAP-INSERT، واكتب شبه الرماز اللازم. (لمسيح: نفّذ إجراء الإدراج المعتاد في شجرة البحث الثنائية، ثم أحر بعض الدورانات لاستعادة خاصية ترتيب الكومة وفق الأصغر.)

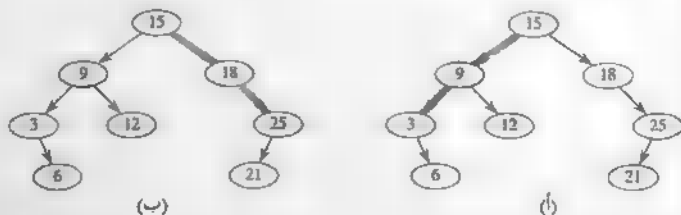
ث. بيّن أن زمن التنفيذ المتوقع لـ TREAP-INSERT هو  $O(\lg n)$ .

بنفّذ TREAP-INSERT بحثاً ثم متالية من الدورانات. ومع أن هاتين العمليتين لهما نفس الزمن المتوقع للتنفيذ، فإنّ لهما عملياً تكلفة مختلفة. فالبحث يقرأ معلومات من الشجرة المكثومة دون تعديلها، على حين يغيّر الدوران مؤشرات الأب والابن ضمن الشجرة المكثومة. وبلاحظ أنه في معظم الحواسيب، تكون عمليات القراءة أسرع بكثير من عمليات الكتابة، لذا فنحن نريد أن يؤدي TREAP-INSERT دورانات أقل. سنبيّن أن العدد المتوقع للدورانات محدود بنهايت.

لإجراء ذلك، نحتاج إلى بعض التعاريف الموضّحة في الشكل 11.13. **العصب الأيسر** *left spine* في شجرة بحث ثنائية  $T$  هو المسار البسيط من الجذر إلى العقدة ذات المفتاح الأصغر. ويتميز آخر، **العصب الأيسر** هو المسار البسيط المؤلف من وصلات يسرى فقط انطلاقاً من الجذر. وبالتناظر، فإن **العصب الأيمن**



**الشكل 10.13** تنفيذ TREAP-INSERT. (أ) الشجرة للكومة الأصلية قبل الإدراج. (ب) الشجرة المكومة بعد إدراج عقدة مفتاحها  $C$  وأفضليتها 25. (ت)-(ث) مراحل وسطية عند إدراج عقدة مفتاحها  $D$  وأفضليتها 9. (ج) الشجرة المكومة بعد إدراج الجزأين (ت) و (ث). (ح) الشجرة للكومة بعد إدراج عقدة مفتاحها  $F$  وأفضليتها 2.



**الشكل 11.13** أعصاب شجرة بحث ثنائية. العصب الأيسر هو للظل في (أ) والعصب الأيمن هو للظل في (ب).

**right spine** هو المسار البسيط المؤلف من وصلات يمتد فقط انطلاقاً من الجذر. طول العصب هو عدد العقد التي يحتويها.

ج. خذ الشجرة المكونة  $T$  مباشرة بعد أن يُدرج TREAP-INSERT العقدة  $x$ . وليكن  $C$  طول العصب الأيمن للشجرة الفرعية اليسرى لـ  $x$ . وليكن  $D$  هو طول العصب الأيسر للشجرة الفرعية اليمنى لـ  $x$ . أثبت أن العدد الكلي للدورانات المنفذة خلال إدراج  $x$  يساوي  $C + D$ .

سنحسب الآن القيم المتوقعة لكل من  $C$  و  $D$ . نفترض، دون فقد العمومية، أن المفاتيح هي  $1, 2, \dots, n$ ، لأننا نقارن أحدهما بالآخر فقط.

ليكن  $k = x.key$  و  $i = y.key$  لكل عقدتين  $x$  و  $y$ ، حيث  $y \neq x$ . نعرف متحولات عشوائية مؤشرة دلالية

$X_{ik} = I\{y \text{ is in the right spine of the left subtree of } x\}$ .

ح. بَيِّنْ أنَّ  $X_{ik} = 1$  إذا وفقط إذا كانت  $y.key < x.key$  و  $y.priority > x.priority$  وأن  $y.priority > z.priority$  لجميع  $z$  التي تحقق  $y.key < z.key < x.key$ .

خ. أثبت أنَّ

$$\begin{aligned} \Pr\{X_{ik} = 1\} &= \frac{(k-i-1)!}{(k-i+1)!} \\ &= \frac{1}{(k-i+1)(k-i)} . \end{aligned}$$

د. أثبت أنَّ

$$\begin{aligned} E[C] &= \sum_{j=1}^{k-1} \frac{1}{j(j+1)} \\ &= 1 - \frac{1}{k} . \end{aligned}$$

ذ. استخدم حجة التناظر لتبين أنَّ

$$E[D] = 1 - \frac{1}{n-k+1} .$$

ر. استنتج أنَّ العدد المتوقع للدورانات المنفذة عند إدراج عقدة في شجرة مكونة أصغر من 2.

## ملاحظات الفصل

تعود فكرة توازن شجرة بحث إلى Landis و Adel'son-Vel'ski [2]، اللذين أدخلوا صنفًا من أشجار البحث المتوازنة المسماة "أشجار AVL" في عام 1962، وللوصفة في المسألة 13-3. ثم أدخل J. E. Hopcroft صنفًا آخر من أشجار البحث عام 1970 يُسمى "أشجار 2-3" (غير منشورة). تحافظ أشجار 2-3 على التوازن بتناول درجات العقد في الشجرة. يدرس الفصل الثامن عشر تعميمًا للأشجار 2-3 ابتدعه باير وماكريت Bayer و McCreight [35] يسمى الأشجار المعشمة B-trees.

ابتكر باير Bayer [34] الأشجار الحمراء-السوداء تحت اسم "الأشجار المعشمة الثنائية المتناظرة" "symmetric binary B-trees". ودرس غوياس وسيدغويك Guibas و Sedgewick [155] خصائصها مطولاً وأدخلوا اصطلاح اللون أحمر/أسود. في حين أعطى أندرسون Andersson [15] نوعًا آخر من الأشجار الحمراء-السوداء أكثر سهولة في الترجمة. يسمى غايس Weiss [351] هذا النوع AA-trees. وهو يشبه الأشجار الحمراء-السوداء إلا أنَّ الأبناء اليُسرى قد لا يكونون حمراء أبدًا.

اقترح سيدل وأراغون Seidel و Aragon [309] الأشجار للكمومة treaps، موضوع المسألة 13-4، وهي التمييز المفضل لمجموع في LEDA [253]، وهي مجموعة منخزة جيدًا من بنى المعطيات والخوارزميات. وهناك أنواع أخرى كثيرة من أشجار البحث المتوازنة، منها الأشجار المتوازنة في الثقل [264]، والأشجار ذات الـ  $k$  جارات  $k$ -neighbor trees [245]، وأشجار scapegoat [127]. ولعلَّ أكثرها إثارة للاهتمام أشجار سبلاي "splay trees" التي أدخلها سليتور ونارجان Sleator و Tarjan [320]، وهي ذاتية التصحيح (انظر Tarjan [330] للوقوف على توصيف جيد لهذه الأشجار). تحافظ أشجار سبلاي على التوازن دون شرط توازن صريح مثل اللون. بل إنَّ العمليات (التي تتضمن دورانات) تُنفَّذ ضمن الشجرة كلما جرى النفاذ إليها. التكلفة الماحدة (انظر الفصل السابع عشر) لكل عملية على شجرة ذات  $n$  عقدة هي  $O(\lg n)$ .

توفّر لوائح سكيب Skip lists [286] بديلاً للأشجار الثنائية المتوازنة. وهي لوائح مترابطة مزوّدة بعدد من المؤشرات الإضافية. تُنفَّذ كل عملية معجمية على لاتحة سكيب من  $n$  عنصرًا في زمن  $O(\lg n)$ .



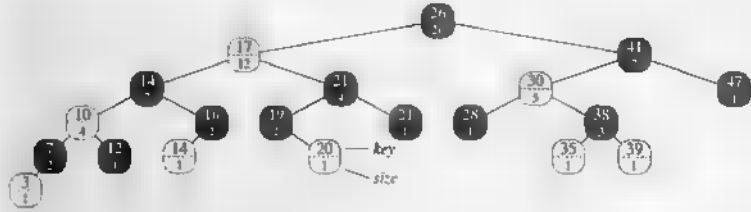
## 14 إغناء بنى المعطيات

لا تحتاج بعض الحالات الهندسية إلى أكثر من بنية معطيات في "كتاب مدرسي" - من مثل لائحة مضاعفة الترابط، أو جدول تليد، أو شجرة بحث ثنائية - غير أن العديد من الحالات الأخرى تحتاج إلى شيء من الإبداع. ومع ذلك، قد تحتاج في حالات نادرة إلى ابتكار غط جديد بالكامل من بنى المعطيات. أما في الأغلب الأعم، فيكفي أن نفني بنى المعطيات الموجودة في الكتب الدراسية بتزويدها بمعلومات إضافية. يمكنك بعدها أن ترمج عمليات جديدة على بنى المعطيات لدعم التطبيق المرغوب فيه. على أن إغناء بنى المعطيات لا يكون مباشرًا دومًا، بل يجب أن تُجرى العمليات التقليدية على بنى المعطيات بتحديث المعلومات المضافة والمحافظة عليها.

يناقش هذا الفصل ببنى معطيات أنشأناها بإغناء الأشجار الحمراء-السوداء. يصف المقطع 1.14 بنية معطيات تدعم عمليات إحصائيات الترتيب العامة في مجموعة ديناميكية، بحيث يمكننا أن نجد بسرعة العدد الأصغر ذا الترتيب  $i$  في مجموعة، أو مرتبة عنصر معطى ضمن الترتيب الكلي لعناصر المجموعة. ويليخص المقطع 2.14 إجراءات إغناء بنية معطيات، ويقدم مرهنة يمكنها تبسيط عملية إغناء الأشجار الحمراء-السوداء. ويستخدم المقطع 3.14 هذه المرهنة للمساعدة في تصميم بنية معطيات للمحافظة على مجموعة ديناميكية من المجالات، كالمجالات الزمنية. فإذا كان لدينا مجال تريد الاستعلام عنه، يمكننا عندئذ إيجاد المجال الذي يتراكب overlap معه في المجموعة وبسرعة.

### 1.14 إحصائيات الترتيب الديناميكية

مهد الفصل 9 لمفهوم إحصائية الترتيب. وبالأخص، فإن إحصائية الترتيب من الدرجة  $n$  في مجموعة من  $n$  عنصرًا حيث  $\{1, 2, \dots, n\}$  هي ببساطة عنصر المجموعة الذي يمتلك المفتاح الأصغر ذا الترتيب  $i$ . وقد رأينا كيف نحدد أية إحصائية ترتيب في زمن  $O(n)$  انطلاقًا من مجموعة غير مرتبة. وسنرى في هذا المقطع كيف يمكن تعديل الأشجار الحمراء-السوداء بحيث نستطيع تحديد أي إحصائية ترتيب لمجموعة ديناميكية في زمن  $O(\lg n)$ ، وسنرى كذلك كيف يمكن حساب مرتبة  $rank$  عنصر - أي موقعه في الترتيب الخطي للمجموعة - في زمن  $O(\lg n)$ .



**الشكل 1.14** شجرة إحصائيات الترتيب، وهي شجرة حمراء-سوداء مُقَنّاة. العقد المظللة حمراء والعقد الفاتحة سوداء. إضافة إلى الواصفات المألوفة في كل عقدة  $x$ ، جرت إضافة واصفة  $x.size$  وهي تعبر عن عدد العقد-عدا الحارس- في الشجرة الفرعية التي جذرها  $x$ .

يبين الشكل 1.14 بنية معطيات تستطيع أن تدعم عمليات إحصائيات الترتيب السريعة. وتُعرف شجرة إحصائيات الترتيب  $T$  order-statistic tree بأنها شجرة حمراء-سوداء مزوّدة بمعلومات إضافية مخزنة في كل عقدة. فإلى جانب الواصفات المألوفة في الأشجار الحمراء-السوداء وهي  $x.key$  و  $x.color$  و  $x.p$  و  $x.left$  و  $x.right$  الموجودة في كل عقدة  $x$ ، لدينا الواصفة  $x.size$ . تتضمن هذا الواصفة عدد العقد (الداخلية) في الشجرة الفرعية التي جذرها  $x$  (ومنها  $x$  نفسها)، أي أنها تتضمن حجم الشجرة الفرعية. فإذا عرفنا حجم الحارس بأنه 0، أي إذا جعلنا  $T.nil.size$  مساوية 0، عندها تكون لدينا المساواة:

$$x.size = x.left.size + x.right.size + 1.$$

ليس من الضروري أن تكون للمفاتيح تمايزية في شجرة إحصائيات الترتيب (فالشجرة الواردة في الشكل 1.14، مثلاً، لها مفتاحان قيمتهما 14 ومفتاحان قيمتهما 21). في حال وجود مفاتيح متساوية لا يكون مفهوم المرتبة المذكور أنفاً معرّفًا تعريفًا جيّدًا، ونزيل هذا الالتباس في شجرة إحصائيات الترتيب بتعريف مرتبة عنصر على أنها الموقع الذي سيظهر فيه فيما لو طبعنا الشجرة بتحوال بيني inorder walk. في الشكل 1.14، مثلاً، تكون مرتبة المفتاح 14 المخزن في عقدة سوداء هي 5، وتكون مرتبة المفتاح 14 المخزن في عقدة حمراء هي 6.

#### استحضار عنصر ذي مرتبة معطاة

قبل أن نبين كيف نحافظ على معلومة الحجم هذه أثناء الإدراج والحذف، لننتهض لتحيز استغلّتين لإحصائيات الترتيب يستخدمان هذه المعلومة الإضافية. نبدأ بعملية استحضار عنصرًا ذا مرتبة معطاة. يُعيد الإجراء  $OS-SELECT(x, i)$  مؤشرًا إلى العقدة التي تحتوي للمفتاح الأصغر ذا الترتيب  $i$  في الشجرة الفرعية التي رأسها  $x$ . ولإيجاد العقدة ذات للمفتاح الأصغر ذي الترتيب  $i$  في شجرة إحصائيات الترتيب  $T$  نستدعي  $OS-SELECT(T.root, i)$ .

```

OS-SELECT(x, i)
1  r = x.left.size + 1
2  if i == r
3      return x
4  elseif i < r
5      return OS-SELECT(x.left, i)
6  else return OS-SELECT(x.right, i - r)

```

في السطر 1 من OS-SELECT نحسب  $r$  وهو مرتبة العقدة  $x$  في الشجرة الفرعية التي جذرها  $x$ . إن قيمة  $x.left.size$  هي عدد العقد التي ترد قبل  $x$  في تحوّل بيني للشجرة الفرعية التي جذرها  $x$ . وهكذا يكون  $x.left.size + 1$  مرتبة  $x$  في الشجرة الفرعية التي جذرها  $x$ . إذا كان  $i = r$  فإن العقدة  $x$  تكون العنصر الأصغر ذا الترتيب  $i$ ، وعليه فإننا نعيد  $x$  في السطر 3. وإذا كان  $i < r$  يكون العنصر الأصغر ذو الترتيب  $i$  موجوداً في الشجرة الفرعية اليسرى لـ  $x$ ، لذا نطبق العودية على  $x.left$  في السطر 5. وإذا كان  $i > r$  يكون العنصر الأصغر ذو الترتيب  $i$  موجوداً في الشجرة الفرعية اليمنى لـ  $x$ . ومما كانت الشجرة الفرعية التي جذرها  $x$  تحتوي  $r$  عنصراً ترد قبل الشجرة الفرعية اليمنى لـ  $x$  في التحوّل بيني للشجرة، فإن العنصر الأصغر ذا الترتيب  $i$  في الشجرة الفرعية التي جذرها  $x$  يكون هو العنصر الأصغر ذا الترتيب  $(i - r)$  في الشجرة الفرعية التي جذرها  $x.right$ . يحدّد هذا العنصر عودياً في السطر 6.

ولمعالجة كيفية عمل OS-SELECT، نبحث عن العنصر الأصغر ذي الترتيب 17 في شجرة إحصائيات الترتيب الواردة في الشكل 1.14. نبدأ بالجذر  $x$  الذي يعمل المفتاح 26 ولدينا قيمة  $i = 17$ . ومما كان حجم الشجرة الفرعية اليسرى لـ 26 هو 12، فإن مرتبتها هي 13. وهكذا، نعلم أن العقدة ذات المرتبة 17 هي  $4 = 17 - 13$  أي إننا نبحث عن العنصر الرابع في الصفر في الشجرة الفرعية اليمنى لـ 26. بعد الاستدعاء العودي تصبح  $x$  هي العقدة ذات المفتاح 41 و  $i = 4$ . ومما كان حجم الشجرة الفرعية اليسرى لـ 41 هو 5، فإن رتبته في شجرته الفرعية هي 6. وهكذا نعلم أن العقدة ذات المرتبة 4 هي العنصر الرابع في الصفر في الشجرة الفرعية اليسرى لـ 41. وبعد الاستدعاء العودي تصبح  $x$  هي العقدة ذات المفتاح 30 ومرتبتها في شجرتها الفرعية هو 2. ومن ثمّ، نطبق العودية مرة ثانية لنجد العنصر الأصغر الثاني ( $4 - 2 = 2$ ) في الشجرة الفرعية التي جذرها هو العقدة ذات المفتاح 38. هنا، نجد أن حجم الشجرة الفرعية اليسرى لهذه العقدة هو 1، وهذا يعني أنها العنصر الثاني في الصفر. وهكذا يعيد الإجراء مؤشراً إلى العقدة ذات المفتاح 38.

ولما كان كل استدعاء عودي يقودنا في كل مرة إلى مستوى أعمق واحد داخل شجرة إحصائيات الترتيب، فإن الزمن الإجمالي لـ OS-SELECT يتناسب في أسوأ الحالات مع ارتفاع الشجرة. وحيث إن الشجرة هي شجرة حمراء-سوداء، فإن ارتفاعها هو  $O(\lg n)$  حيث  $n$  هو عدد العقد. وهكذا يكون زمن تنفيذ OS-SELECT هو  $O(\lg n)$  في حالة مجموعة ديناميكية ذات  $n$  عنصراً.

## تحديد مرتبة عنصر

ليكن لدينا مؤشر إلى عقدة  $x$  في شجرة إحصائيات الترتيب  $T$ . بعد الإجراء OS-RANK موقع  $x$  في الترتيب الخطي الذي يحدده التجوال البيني للشجرة  $T$ .

OS-RANK( $T, x$ )

```

1   $r = x.left.size + 1$ 
2   $y = x$ 
3  while  $y \neq T.root$ 
4      if  $y == y.p.right$ 
5           $r = r + y.p.left.size + 1$ 
6       $y = y.p$ 
7  return  $r$ 
```

يعمل الإجراء كما يلي: يمكن النظر إلى مرتبة  $x$  على أنها عدد العقد التي تسبق  $x$  في تجوال بيني للشجرة مضافاً إليه 1 لـ  $x$  نفسه. يحافظ OS-RANK على لامتغير الحلقة التالي:

عند بداية كل تكرار للحلقة while في الأسطر 3-6، يكون  $r$  هو مرتبة  $x.key$  في الشجرة الفرعية التي جذرها العقدة  $y$ .

نستخدم لامتغير الحلقة هنا لنبين أن OS-RANK يعمل بوجه صحيح كما يلي:

الاستدعاء: قبل التكرار الأول، يضع السطر الأول في  $r$  مرتبة  $x.key$  في الشجرة الفرعية التي جذرها  $x$ . إن وضع  $x = y$  في السطر 2 يجعل لامتغير الحلقة صحيحاً عندما بنفذ الاختبار في السطر 3 أول مرة.

المحافظة: في نهاية كل تكرار للحلقة while، نجعل  $y = y.p$ . لذا، يجب أن نبين أنه إذا كان  $r$  هو مرتبة  $x.key$  في الشجرة الفرعية التي جذرها  $y$  في بداية جسم الحلقة، فإن  $r$  يكون مرتبة  $x.key$  في الشجرة الفرعية التي جذرها  $y.p$  في نهاية جسم الحلقة. وفي كل تكرار للحلقة while نتم بالشجرة الفرعية التي جذرها  $y.p$ . وقد سبق أن حسبنا عدد العقد في الشجرة الفرعية التي جذرها هو العقدة  $y$  التي تسبق  $x$  في التجوال البيني، لذا، يجب إضافة العقد في الشجرة الفرعية التي جذرها هو أخ  $y$  الذي يسبق  $x$  في التجوال البيني ويضاف 1 أيضاً لـ  $y.p$  إذا كان هو بدوره يسبق  $x$ . أما إذا كان  $y$  ابناً يسر، عندها لا يمكن لـ  $y.p$  ولا لأية عقدة في الشجرة الفرعية اليمنى لـ  $y.p$  أن تسبق العقدة  $x$ . عندها نترك  $r$  وحدها. وفيما عدا ذلك، فإن  $y$  ابن أيمن، وجميع العقد في الشجرة الفرعية اليسرى لـ  $y.p$  تسبق  $x$ . كما هو حال  $y.p$  نفسه. لذا، نضيف  $y.p.left.size + 1$ ، في السطر 5، إلى القيمة الحالية لـ  $r$ .

الانتهاء: تنتهي الحلقة عندما تصبح  $y = T.root$  بحيث تكون الشجرة الفرعية التي جذرها  $y$  هي كامل الشجرة. لذا، تكون قيمة  $r$  هي مرتبة  $x.key$  في كامل الشجرة.

مثلاً، إذا نفذنا OS-RANK على شجرة إحصائيات الترتيب في الشكل 1.14 للبحث عن مرتبة العقدة ذات المفتاح 38، حصلنا على المتتالية اللاحقة من قيم  $y.key$  و  $r$  في بداية الحلقة **while**:

التكرار	$y.key$	$r$
1	38	2
2	30	4
3	41	4
4	26	17

يعيد الإجراء المرتبة 17.

ولما كان كل تكرار للحلقة **while** يستغرق  $O(1)$  من الزمن، و  $y$  ترتفع نحو الأعلى بمقدار مستوى واحد في الشجرة مع كل تكرار، فإن زمن تنفيذ OS-RANK يتناسب في أسوأ الحالات مع ارتفاع الشجرة:  $O(\lg n)$  في شجرة إحصائيات الترتيب ذات  $n$  عقدة.

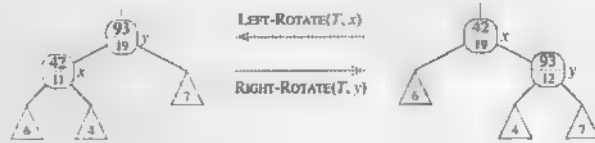
### المحافظة على أحجام الأشجار الفرعية

إذا عُلمت قيمة الواسعة  $size$  في كل عقدة، استطاع OS-SELECT و OS-RANK حساب معلومات إحصائية الترتيب بسرعة. ولكن إذا لم تستطع عمليات التعديل الأساسية على الأشجار الحمراء-السوداء المحافظة على هذه الواسعات محافظة فعالة فستذهب جهودنا سُدى. وسنبين الآن أن عمليتي الإدراج والحذف تحافظان على أحجام الأشجار الفرعية دون التأثير في الزمن المقارب لتنفيذ أي من هاتين العمليتين.

لاحظنا في المقطع 3.13 أن الإدراج في شجرة حمراء-سوداء يتألف من مرحلتين. تعتمد المرحلة الأولى على التزول في الشجرة اعتباراً من الجذر، وإدراج العقدة الجديدة كابت لعقدة موجودة. في حين تقوم المرحلة الثانية بالصعود إلى الأعلى في الشجرة، وتغيير الألوان وإجراء الدورانات اللازمة للمحافظة على الخصائص الحمراء-السوداء.

وللمحافظة على أحجام الأشجار الفرعية في المرحلة الأولى، ما علينا إلا إضافة 1 إلى  $x.size$  في كل عقدة  $x$  على المسار البسيط للمعد من الجذر نزولاً نحو الأوراق. تحصل العقدة المضافة على قيمة 1 في واصفة  $size$  الخاصة بها. ولما كان ثمة  $O(\lg n)$  عقدة على المسار المذكور، فإن التكلفة الإضافية للمحافظة على الواسعات  $size$  تكون  $O(\lg n)$ .

أما في المرحلة الثانية، فتمثل التغييرات البتوية الوحيدة على الشجرة الحمراء-السوداء الأساسية في الدورانات التي يحدث منها على الأكثر اثنان. وكذلك فإن الدوران هو عملية محلية: إذ إن عقدتين فقط تصبح واصفة  $size$  فيهما غير صحيحة. والوصلة التي ينقذ حولها الدوران تتوقف على هاتين العقدتين. وبالرجوع إلى رمز  $LEFT-ROTATE(T, x)$  الوارد في المقطع 2.13 نضيف السطرين التاليين:



**الشكل 2.14** تحديث أحجام الأشجار الفرعية أثناء الدورانات. إن الوصلة التي يجري الدوران حولها منوطة بالعقدتين اللتين يجب تحديث واصفات size فيهما. وإن عمليات التحديث عملية، ونحتاج نقط إلى المعلومة size المعزونة في  $x$  و  $y$  وإلى جذور الأشجار الفرعية لتمثلة على شكل مثلثات.

13  $y.size = x.size$

14  $x.size = x.left.size + x.right.size + 1$

يوضح الشكل 2.14 كيف تُحدث هاتان الوصفتان. ويكون التغير في RIGHT-ROTATE متماثلًا لهذا التغير. وحيث إنه سَيُنفَّذ دورانان على الأكثر أثناء الإدراج في شجرة حمراء-سوداء، فإننا نحتاج إلى زمن إضافي  $O(1)$  فقط كي نُحدث الواصفات size في المرحلة الثانية. وهكذا، يكون الزمن الإجمالي للإدراج في شجرة إحصائية الترتيب ذات  $n$  عقدة هو  $O(\lg n)$ ، وهو مقارب للإدراج في شجرة حمراء-سوداء عادية.

وبالمثل، يتألف الحذف من شجرة حمراء-سوداء مرحلتين: المرحلة الأولى تعمل على شجرة البحث الأساسية، والثانية تتسبب في ثلاثة دورانات على الأكثر، وفيما عدا ذلك فهي لا تقوم بأية تغييرات بنيوية (انظر المقطع 4.13). فالمرحلة الأولى إما أن تُحذف عقدة واحدة  $y$  من الشجرة، وإما أن تنتقل إلى موقع أعلى منها ضمن الشجرة. ولتحديث أحجام الأشجار الفرعية نقوم ببساطة بعبور المسار من العقدة  $y$  (اعتبارًا من موضعها الأصلي في الشجرة) إلى الأعلى باتجاه الجذر، ونقص 1 من الوصفة size لكل عقدة على المسار. ولما كان طول هذا المسار هو  $O(\lg n)$  في حالة شجرة حمراء-سوداء ذات  $n$  عقدة، فإن الزمن الإضافي اللازم للمحافظة على الواصفات size في المرحلة الأولى هو  $O(\lg n)$ . أما الدورانات ذات الزمن  $O(1)$  في المرحلة الثانية من الحذف، فيمكن معالجتها بالطريقة نفسها كما في الإدراج. وهكذا، تستغرق عمليتا الإدراج والحذف، وما يتضمنانه من محافظة على الواصفات size، زمنًا  $O(\lg n)$  في حالة شجرة إحصائيات الترتيب ذات  $n$  عقدة.

## تمارين

### 1-1.14

يَبَيِّن كيف تعمل OS-SELECT( $T.root, 10$ ) على الشجرة الحمراء-السوداء  $T$  التي مرت معنا في الشكل 1.14.

### 2-1.14

يُبين كيف تعمل  $OS-RANK(T, x)$  على الشجرة الحمراء-السوداء  $T$  التي مرت معنا في الشكل 1.14 والعقدة  $x$  حيث  $x.key = 35$ .

### 3-1.14

اكتب نسخة غير عودية من  $OS-SELECT$ .

### 4-1.14

اكتب إجراء عودياً  $OS-KEY-RANK(T, k)$  يتخذ شجرة إحصائية الترتيب  $T$  ومفتاحاً  $k$  دخلاً له، ويعيد مرتبة  $k$  في المجموعة الديناميكية المثلثة بـ  $T$ . افترض أن مفاتيح  $T$  متميزة.

### 5-1.14

لدينا عنصر  $x$  معطى في شجرة إحصائيات الترتيب ذات  $n$  عقدة وعدد طبيعي  $i$ . كيف يمكننا تحديد العنصر اللاحق لـ  $x$  ذي الترتيب  $i$  في الترتيب الخطي للشجرة وذلك خلال زمن  $O(\lg n)$ ؟

### 6-1.14

لاحظ أنه كلما أشرنا إلى الوصفة  $size$  لعقدة ما، سواء في  $OS-SELECT$  أو في  $OS-RANK$ ، فإننا نستعملها لحساب مرتبة عقدة فقط. بناءً على ذلك، افترض أننا نخزن في كل عقدة مرتبتها في الشجرة الفرعية التي تمثل العقدة جذراً لها. يُبين كيف يمكن المحافظة على هذه المعلومة أثناء الإدراج والحذف. (تذكر أن هاتين العمليتين قد تُسببان دورانات.)

### 7-1.14

يُبين كيف تُستخدم شجرة إحصائية الترتيب لإحصاء عدد عمليات الثقلب (انظر المسألة 4-2) في مصفوفة طولها  $n$  في زمن  $O(n \lg n)$ .

### 8-1.14 \*

ليكن لدينا  $n$  نقطة على دائرة بحيث يُعرف كل وتر بنقطتي الطرفين. صِف خوارزمية مقدار زمنها  $O(n \lg n)$  لتحديد عدد أزواج الأوتار التي تتقاطع داخل الدائرة. (مثلاً إذا كانت جميع الأوتار التي عددها  $n$  أفطاراً لنقطة في المركز، فعندها يكون الجواب الصحيح  $\binom{n}{2}$ .) افترض أنه لا يوجد وتران يشتركان في إحدى النهايتين.

## 2.14 كيف نغني بنية معطيات

إن الفرض من عملية إغناء بنية معطيات أساسية هو تعزيز فعالية إضافية، وتحدث هذه العملية حدوداً متكرراً عند تصميم الخوارزميات. سنستخدم هذه العملية أيضاً في المقطع التالي لتصميم بنية معطيات تدعم العمليات على المجالات. وسندرس في هذا للمقطع الخطوات المتبعة في عملية الإغناء، وسنبرهن أيضاً نظرية تسمح لنا بإغناء الأشجار الحمراء-السوداء بسهولة في حالات كثيرة.

يمكن تقسيم عملية إغناء بنية معطيات إلى أربع خطوات:

1. اختيار بنية معطيات أساسية.
2. تحديد المعلومات الإضافية المراد المحافظة عليها في بنية للمعطيات الأساسية.
3. التحقق من إمكان المحافظة على المعلومات الإضافية في عمليات التعديل الأساسية على بنية للمعطيات الأساسية.
4. استحداث عمليات جديدة.

وكما في أية طريقة تصميم منهجية، لا يفترض بك أن تتبع الخطوات وفق الترتيب المعطى اتباعاً أعمى؛ فمعظم أعمال التصميم تقوم على عنصر المحاولة والخطأ، ويحدث التطور في جميع الخطوات على التوازي عادةً. فمثلاً، من العيب تحديد معلومات إضافية وابتكار عمليات جديدة (المرحلتان 2 و 4) إذا لم تكن لدينا القدرة على المحافظة على المعلومات الإضافية معافضة فعالة. على كل حال، فإن هذه الطريقة ذات الخطوات الأربع من شأنها أن توجه جهودك توجيهاً جيداً في إغناء بنى للمعطيات، وهي في الوقت نفسه طريقة جيدة لتنظيم التوثيق المتعلق ببنية للمعطيات المُضغاة.

وقد اتبعنا هذه الخطوات في المقطع 1.14 في تصميم أشجار إحصائيات الترتيب. ففي الخطوة الأولى اخترنا أشجاراً حمراء-سوداء كبنية معطيات أساسية. ولعل المر في صلاحية الأشجار الحمراء-السوداء يكمن في دعمها الفعال لعمليات المجموعات الديناميكية الأخرى المتعلقة بالترتيب الكلي مثل MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR.

وفي الخطوة 2، أضفنا الوصفة size التي تخزن فيها كل عقدة  $x$  حجم الشجرة الفرعية التي جذرها  $x$ . وترمي المعلومة الإضافية عموماً إلى جعل العمليات أكثر فعالية. فمثلاً، كان من الممكن تنجز OS-SELECT و OS-RANK باستخدام المفاتيح المخزنة في الشجرة فقط، ولكننا لم نكن نحصل على زمن التنفيذ  $O(\lg n)$ . وفي بعض الأحيان تكون المعلومة الإضافية على شكل مؤشر على معلومات بدلاً من كونها معطيات كما في التمرين 1.14-2.

وفي الخطوة 3، تأكدنا أن الإدراج والحذف قد يحافظان على الوصفات size مع بقاء التنفيذ ضمن زمن  $O(\lg n)$ . في الحالة التالية، نحتاج فقط إلى تحديث عدد قليل من عناصر بنية للمعطيات بهدف المحافظة على المعلومة الإضافية. فمثلاً، لو خزنا ضمن كل عقدة في الشجرة مرتبتها، نُفِّذُ الإجراء OS-SELECT و OS-RANK بسرعة، ولكن إدراج عنصر أصغري جديد قد يتسبب في تغيير هذه المعلومة في كل عقدة من الشجرة. أما لو قمنا بدلاً من ذلك بتخزين حجم الأشجار الفرعية، لأحدث إدراج عنصر جديد تغييراً في المعلومات في  $O(\lg n)$  عقدة فقط.

في الخطوة 4، استحدثنا العمليتين OS-SELECT و OS-RANK. وواقع الأمر أن الحاجة إلى عمليات



جديدة كانت هي الدافع لنا للاهتمام بإغناء بنية المعطيات بالدرجة الأولى، علماً بأننا نستعمل بين حين وآخر للمعلومة الإضافية لتسريع العمليات الموجودة بدلاً من تطوير عمليات جديدة كما في التمرين 1-2.14.

### إغناء الأشجار الحمراء-السوداء

عندما تكون الأشجار الحمراء-السوداء هي البنية الأساسية لبنية معطيات مُنغاة، يمكننا عندها الرهان على أن الإدراج والحذف يستطيعان المحافظة على بعض أنواع المعلومات الإضافية محافضةً فعالةً، وهذا ما يجعل الخطوة 3 سهلة جداً. وبشبه برهان النظرية التالية للنقشة التي وردت معنا في المقطع 1.14، وهي أنه يمكن المحافظة على الوصفة  $size$  في أشجار إحصائيات الترتيب.

#### نظرية 1.14 (إغناء شجرة حمراء-سوداء)

لكن  $f$  واصفة تُغي شجرة حمراء-سوداء  $T$  ذات  $n$  عقدة، ونفترض أن قيمة  $f$  في كل عقدة  $x$  تعتمد فقط على المعلومات الموجودة في العقد  $x$  و  $x.left$  و  $x.right$ . وقد تتضمن  $x.left.f$  و  $x.right.f$ . عندها، يمكن المحافظة على قيم  $f$  في جميع عقد  $T$  أثناء الإدراج والحذف دون التأثير بطريقة مقاربة على الأداء  $O(\lg n)$  لهذه العمليات.

**البرهان** تقوم الفكرة الرئيسة في البرهان على أن التغيير في واصفة  $f$  في عقدة  $x$  ينتقل إلى أسلاف  $x$  فقط في الشجرة، بمعنى أن تغير  $x.f$  قد يضطرنا إلى تحديث  $x.p.f$  فقط دون سواه؛ وقد يضطرنا تحديث  $x.p.f$  بدوره إلى تحديث  $x.p.p.f$  فقط دون سواه؛ وهكذا صعوداً إلى الأعلى في الشجرة. وعندما تُحدث  $T.root.f$ ، فإنه لا توجد عقدة أخرى تتعلق بالقيمة الجديدة، لذا تتوقف العملية. ولما كان ارتفاع الشجرة الحمراء-السوداء هو  $O(\lg n)$ ، فإن تغير واصفة  $f$  في عقدة يكلفنا  $O(\lg n)$  لتحديث جميع العقد المعتمدة على هذا التغيير.

يتألف إدراج عقدة  $x$  في  $T$  من مرحلتين. (انظر المقطع 3.13). تُدرج المرحلة الأولى  $x$  باعتباره ابناً لعقدة موجودة  $x.p$ . ويمكننا حساب  $x.f$  في زمن  $O(1)$  لأنه -بحسب فرضنا- يتعلق فقط بالمعلومات الموجودة في الوصفات الأخرى لـ  $x$  نفسها وبالمعلومات الموجودة في أبناء  $x$ ، إلا أن ابني  $x$  هما الحارس  $T.null$ . بعد حساب  $x.f$  ينتقل التغيير إلى الأعلى في الشجرة، لذا، يكون الزمن الإجمالي للمرحلة الأولى من الإدراج هو  $O(\lg n)$ . وفي أثناء المرحلة الثانية، تأتي التغييرات البنيوية الوحيدة على الشجرة من الدورانات. ولما كان الدوران يتسبب في التغيير على عقدتين فقط، فإن الزمن الإجمالي لتحديث الوصفات  $f$  هو  $O(\lg n)$  لكل دوران. ولما كان عدد الدورانات أثناء الإدراج لا يتجاوز اثنين، فإن الزمن الإجمالي للإدراج يكون  $O(\lg n)$ .

وعلى غرار الإدراج، يتألف الحذف من مرحلتين. (انظر للمقطع 4.13). ففي المرحلة الأولى، تُحدث التغييرات على الشجرة عند إزالة العقدة المحذوفة من الشجرة. فإذا كان للعقدة المحذوفة ابنان في ذلك الوقت،

انتقل خلفها إلى موضع العقدة المحذوفة. يكلف انتقال التحديثات على  $f$  التي تسببت بها هذه التغييرات  $O(\lg n)$  على الأكثر، لأن التغييرات تؤثر على الشجرة عملياً. ويتطلب تصليح الشجرة الحمراء-السوداء أثناء المرحلة الثانية ثلاثة دورانات على الأكثر، ويحتاج كل دوران إلى زمن  $O(\lg n)$  على الأكثر لانتقال التحديثات على  $f$ . لذا، وكما في الإدراج، يكون الزمن الإجمالي للحذف  $O(\lg n)$ . ■

وفي حالات كثيرة، من مثل المحافظة على الواصفات  $size$  في أشجار إحصائيات الترتيب، تكون تكلفة التحديث بعد الدوران هي  $O(1)$  بدلاً من  $O(\lg n)$  للمستلمة من برهان النظرية 1.14. يعطي التمرين 3-2.14 مثالاً عن ذلك.

### تمارين

#### 1-2.14

يُرى كيف يمكننا، بإضافة المؤشرات إلى العقد، دعم كلٍّ من الاستعلامات MAXIMUM و MINIMUM و SUCCESSOR و PREDECESSOR في المجموعات الديناميكية خلال زمن  $O(1)$  في أسوأ الحالات في شجرة إحصائيات الترتيب المُفَنَدة. ينبغي ألا يتأثر الأداء للمقارب لبقية العمليات الحمراء على أشجار إحصائيات الترتيب.

#### 2-2.14

هل يمكننا المحافظة على الارتفاعات السوداء للعقد في شجرة حمراء-سوداء على اعتبار أنها واصفات في عقد الشجرة دون التأثير في الأداء للمقارب لأيٍّ من العمليات على الشجرة الحمراء-السوداء؟ يَرى كيف يمكن ذلك أو أثبت لم لا يمكن ذلك. ماذا عن المحافظة على أعمال العقد؟

#### \* 3-2.14

ليكن  $\otimes$  معاملاً ثنائيًا تجميعيًا، وليكن  $a$  واصفةً محفوظةً في كل عقدة من عقد شجرة حمراء-سوداء. ولنفترض أننا نريد أن نضخّن في كل عقدة  $x$  واصفةً إضافيةً  $f$  بحيث يكون  $x.f = x_1 \cdot a \otimes x_2 \cdot a \otimes \dots \otimes x_m \cdot a$  حيث  $x_1, x_2, \dots, x_m$  هي السرد البيني للعقد في الشجرة الفرعية التي جذرها  $x$ . يَرى كيف يمكن تحديث الواصفات  $f$  في زمن  $O(1)$  بعد إجراء دوران. عدّل برهانك قليلاً بحيث يمكن تطبيقه على الواصفات  $size$  في أشجار إحصائيات الترتيب.

#### \* 4-2.14

نرغب في إغناء أشجار حمراء-سوداء بالعملية  $RB-ENUMERATE(x, a, b)$  التي تعطي خرجاً يتمثل في جميع المفاتيح  $k$  حيث  $a \leq k \leq b$  في شجرة حمراء-سوداء جذرها  $x$ . صِفْ كيف يمكن تنحيز  $RB-ENUMERATE$  في زمن  $\Theta(m + \lg n)$ ، حيث  $m$  عدد المفاتيح في المخرج و  $n$  عدد العقد الداخلية في الشجرة. (تلميح: لا حاجة إلى إضافة واصفات جديدة إلى الشجرة الحمراء-السوداء.)

## 3.14 أشجار المجالات

في هذا المقطع، سنُغني الأشجار الحمراء-السوداء بحيث تدعم العمليات على المجموعات الديناميكية الخاصة بالمجالات. يعرف المجال المغلق *closed interval* بأنه زوج مرتب من الأعداد الحقيقية  $[t_1, t_2]$  حيث  $t_1 \leq t_2$ . يمثل المجال  $[t_1, t_2]$  المجموعة  $\{t \in \mathbb{R} : t_1 \leq t \leq t_2\}$ . أما المجالات *المفتوحة* *open* و *نصف المفتوحة* *half-open* فلا تتضمن كلتا النهايتين أو إحداها في المجموعة على الترتيب. سنفترض في هذا المقطع أن المجالات مغلقة، وبذلك يكون توسيع النتائج إلى المجالات المفتوحة ونصف المفتوحة أمراً مباشراً وواضحاً على مستوى المفاهيم.

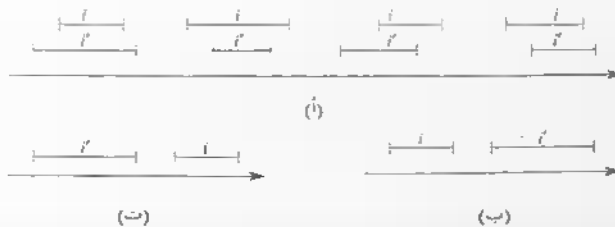
هذا وإن استعمال المجالات ملائم لتمثيل الأحداث التي يشغل كل منها مدة مستمرة. فقد نرغب مثلاً بالاستعلام من قاعدة معطيات عن المجالات الزمنية لمعرفة الأحداث التي تجري أثناء مجال مُعطى. تقدّم بنية المعطيات في هذا المقطع وسيلة ناعمة للمحافظة على قاعدة معطيات المجالات هذه.

يمكن أن نمثل المجال  $[t_1, t_2]$  كفرض  $i$  مزوّد بالوصفتين  $i.low = t_1$  (النقطة الطرفية الأدنى *low endpoint*) و  $i.high = t_2$  (النقطة الطرفية العليا *high endpoint*). نقول إن المجالين  $i$  و  $i'$  متراكبان *overlap* إذا كان  $i \cap i' \neq \emptyset$ ، بمعنى أن  $i.low \leq i'.high$  و  $i'.low \leq i.high$ . وكما يبيّن الشكل 3.14، يخفّق أيّ زوج من المجالات  $i$  و  $i'$  خاصية *النفّث الثلاثي للمجالات* *interval trichotomy* بمعنى أحصا بمقتضى واحدة فقط من هذه الخصائص الثلاث التالية:

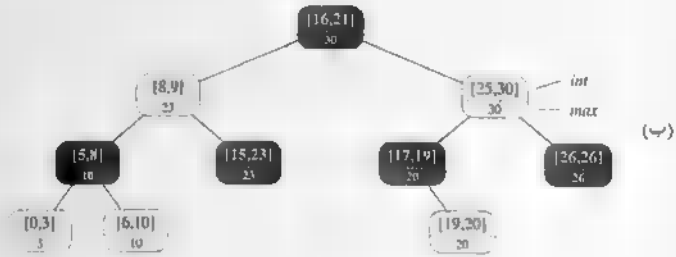
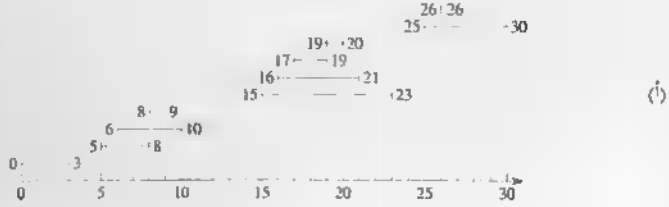
أ.  $i$  و  $i'$  يتراكبان

ب. يقع  $i$  إلى يسار  $i'$  (أي  $i.high < i'.low$ )

ت. يقع  $i$  إلى يمين  $i'$  (أي  $i'.high < i.low$ )



الشكل 3.14 خاصية النفّث الثلاثي للمجالات في حالة مجالين مغلقين  $i$  و  $i'$ . (أ) إذا تراكب  $i$  و  $i'$  فهناك أربع حالات يكون في كل منها  $i.low \leq i'.high$  و  $i'.low \leq i.high$ . (ب) المجالان لا يتراكبان و  $i'.high < i.low$ . (ت) المجالان لا يتراكبان و  $i.high < i'.low$ .



**الشكل 4.14** شجرة بمحالات. (أ) مجموعة من 10 محالات تظهر مرتبة من الأدنى إلى الأعلى حسب النقطة الطرفية اليسرى. (ب) الشجرة التي تمثل هذه المحالات. تحتوي كل عقدة  $\square$  بمحالا يظهر على الشكل فوق الخط المنقط، والقيمة العظمى لأية نقطة طرفية للمحالات في الشجرة الفرعية التي جذرها  $x$ ، وتظهر في الشكل تحت الخط المنقط. يسمح التحوال البيني للشجرة بمرء العقد مرتبة حسب النقطة الطرفية اليسرى.

تعرّف **شجرة المجال** *interval tree* بأنها شجرة حراء-سوداء تحافظ على مجموعة ديناميكية من العناصر التي يحتوي كل عنصر  $x$  فيها بمحالا  $x.int$ . تدعم أشجار المجال العمليات التالية:

**INTERVAL-INSERT( $T, x$ )** وهي تضيف إلى شجرة المجال  $T$  العنصر  $x$  الذي يفترض أن تحتوي الوصفة  $int$  فيه بمحالا.

**INTERVAL-DELETE( $T, x$ )** وهي تحذف العنصر  $x$  من شجرة المجال  $T$ .

**INTERVAL-SEARCH( $T, i$ )** وهي تعيد مؤشرًا إلى العنصر  $x$  الموجود في شجرة المجال  $T$  بحيث يتراكب مجاله مع المجال  $i$ ، أو تعيد مؤشرًا إلى الحارس  $T.nil$  إن لم يوجد هذا العنصر في المجموعة.

يبين الشكل 4.14 كيف تمثل شجرة المجال مجموعة من المحالات. مستنتج الخطوات الأربع في الطريقة التي وصفناها في المقطع 2.14 في سياق مراجعتنا لتصميم شجرة المجال والعمليات التي تنفذ عليها.

### الخطوة 1: بنية المعطيات الأساسية

نختار شجرة حمراء-سوداء بحيث تتضمن كل عقدة  $x$  فيها مجالاً  $x.int$  وبحيث يمثل مفتاح  $x$  النقطة الطرفية الدنيا للمجال  $x.int.low$ . وهكذا، يسرد التحوّل الييني في شجرة بنية المعطيات المجالات مرتبة حسب نقاطها الطرفية الدنيا.

### الخطوة 2: المعلومات الإضافية

إضافة إلى المجالات نفسها، تحتوي كل عقدة  $x$  قيمة  $x.max$ ، تمثل القيمة العظمى لأية نقطة طرفية للمجالات المخزنة في الشجرة الفرعية التي جذرها  $x$ .

### الخطوة 3: المحافظة على المعلومات

يجب أن نتحقق من أن عمليتي الإدراج والحذف تُنجزان في زمن  $O(\lg n)$  في شجرة مجالات ذات  $n$  عقدة. يمكن أن نحدد  $x.max$  بدلالة المجال المعطى  $x.int$  والقيم  $max$  لأبناء العقدة  $x$ :

$$x.max = \max(x.int.high, x.left.max, x.right.max).$$

إذن، وبحسب النظرية 1.14، يتخذ كل من الإدراج والحذف في زمن  $O(\lg n)$ . والواقع أنه بإمكاننا تحديث الواصفات  $max$  بعد إجراء دوران خلال زمن  $O(1)$ ، كما يبين التمرينات 2.14 و 3.14-1.

### الخطوة 4: استحداث عمليات جديدة

إن العملية الجديدة الوحيدة التي نحتاج إليها هي  $INTERVAL-SEARCH(T, i)$  التي تبحث عن عقدة في شجرة  $T$  بحيث يتراكب مجالها مع المجال  $i$ . إذا لم تجد هذه العملية أي مجال يتراكب مع  $i$  في الشجرة فإنها تعيد مؤشراً للحارس  $T.nil$ .

$INTERVAL-SEARCH(T, i)$

```

1   $x = T.root$ 
2  while  $x \neq T.nil$  and  $i$  does not overlap  $x.int$ 
3      if  $x.left \neq T.nil$  and  $x.left.max \geq i.low$ 
4           $x = x.left$ 
5      else  $x = x.right$ 
6  return  $x$ 
```

يبدأ البحث عن مجال يتراكب مع  $i$  انطلاقاً من  $x$  باعتبارها جذراً للشجرة ويستمر نزولاً. ويتوقف البحث عندما يجد مجالاً متراكباً، أو عندما تشير  $x$  إلى الحارس  $T.nil$ . ونظراً إلى أن كل تكرار من الحلقة الرئيسية يستغرق  $O(1)$  من الزمن، ونظراً إلى أن ارتفاع شجرة حمراء-سوداء ذات  $n$  عقدة هو  $O(\lg n)$ ، فإن

الإجراء INTERVAL-SEARCH يستغرق زمنًا  $O(\lg n)$ .

قبل أن نفصح عن السبب الذي يجعل INTERVAL-SEARCH صحيحًا، لندرس كيفية عمله على شجرة المجال في الشكل 4.14. لنفترض أننا نبحث عن مجال يتراكب مع المجال  $i = [22, 25]$ . نبدأ بالجذر  $x$  الذي يحتوي  $[16, 21]$  ولا يتراكب مع  $i$ . ولما كان  $x.left.max = 23$  أكبر من  $i.low = 22$ ، فإن الحلقة تستمر باعتبار  $x$  الابن الأيسر للجذر—وهي العقدة التي تحتوي  $[8, 9]$ ، الذي لا يتراكب كذلك مع  $i$ . في هذه المرة لدينا  $x.left.max = 10$  وهو أصغر من  $i.low = 22$ ، نستمر الحلقة إذاً مع الابن الأيمن لـ  $x$  بدلاً من  $x$ . إن المجال  $[15, 23]$  المخزن في هذه العقدة يتراكب مع  $i$ ، لذا يعيد الإجراء هذه العقدة.

ولنضرب مثالاً على عملية بحث غير ناجحة. لنفترض أننا نرغب بالبحث عن مجال يتراكب مع  $i = [11, 14]$  في شجرة المجال التي الواردة في الشكل 4.14. نبدأ هنا أيضًا بالجذر  $x$ . ولما كان مجال الجذر وهو  $[16, 21]$  لا يتراكب مع  $i$ ، ونظرًا إلى أن  $x.left.max = 23$  أكبر من  $i.low = 11$ ، فإننا نتجه إلى اليسار نحو العقدة التي تحتوي المجال  $[8, 9]$ . إن المجال  $[8, 9]$  لا يتراكب مع  $i$  و  $x.left.max = 10$  أصغر من  $i.low = 11$ ، لذا نستنتج بميتا. (لاحظ أنه لا يوجد أي مجال في الشجرة الفرعية اليسرى يتراكب مع  $i$ ). إن المجال  $[15, 23]$  لا يتراكب مع  $i$ ، وابنه الأيسر هو  $T.nil$ ، لذا نتجه بميتا من جديد وننتهي الحلقة، ويعيد الإجراء الحارس  $T.nil$ .

لكي تتمكن من معرفة السبب الذي يجعل INTERVAL-SEARCH صحيحًا، يجب أن نعرف لماذا يكفي أن نستقصي مساراتًا جيدًا مطلقًا من الجذر. تكمن الفكرة الأساسية في أنه مهما كانت العقدة  $x$ ، إذا كان  $x.int$  لا يتراكب مع  $i$ ، فإن البحث يتجه دومًا باتجاه سليم: فإذا كان ثمة مجال يتراكب في الشجرة فلا بد من أن يعثر عليه البحث. تنص البرهنة التالية على هذه الخاصية بدقة أكبر.

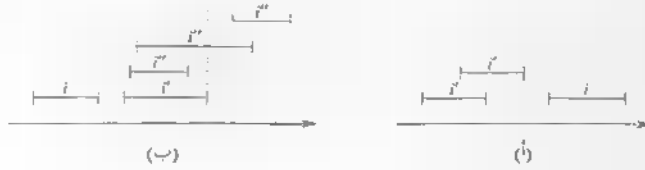
## مبرهنة 2.14

إن تنفيذ الإجراء  $INTERVAL-SEARCH(T, i)$  إما أن يعيد عقدة يتراكب بمجالها مع  $i$ ، وإما أن يعيد  $T.nil$  وعندئذٍ لا تحتوي الشجرة  $T$  على أية عقدة يتراكب بمجالها مع  $i$ .

**البرهان** تنتهي الحلقة **while** الموجودة في الأسطر 5-2 عندما  $x = T.nil$ ، أو عندما يتراكب  $i$  مع  $x.int$ . وفي هذه الحالة الأخيرة، من المؤكد أن إعادة  $x$  هي نتيجة صحيحة. لذا، متركز على الحالة السابقة التي تنتهي فيها الحلقة لأن  $x = T.nil$ .

نستخدم اللامتغير التالي للحلقة **while** في الأسطر 5-2:

إذا كانت الشجرة  $T$  تحتوي مجالاً يتراكب مع  $i$ ، عندها تحتوي الشجرة الفرعية التي جذرها  $x$  مثل هذا المجال.



**الشكل 5.14** الحالات في إثبات المرحلة 2.14. تظهر قيمة  $x.left.max$  في كل حالة على شكل خط منقطع.

(أ) يتبع البحث بحثاً. لا يوجد أي مجال  $i'$  في الشجرة الفرعية اليسرى لـ  $x$  يمكن أن يتراكب مع  $i$ . (ب) يتبع البحث يساراً. تحتوي الشجرة الفرعية اليسرى لـ  $x$  مجالاً يتراكب مع  $i$  (هذه الحالة غير ظاهرة على الشكل)، أو يوجد مجال  $i'$  في الشجرة الفرعية اليسرى لـ  $x$  بحيث  $i'.high = x.left.max$ . لا يزال  $i$  لا يتراكب مع  $i'$ ، ولا يتراكب مع أي مجال  $i''$  في الشجرة الفرعية اليمنى لـ  $x$ ، لأن  $i'.low \leq i''.low$ .

نستخدم لامتغير الحلقة هذا كما يلي:

**الاستدعاء:** قبل التكرار الأول، يجعل السطر 1 العقدة  $x$  جذراً لـ  $T$  بحيث يتحقق اللامتغير.

**المحافظة:** ننفذ كل تكرار للحلقة **while** أحد السطرين 4 أو 5. وسنرى أن لامتغير الحلقة يبقى صحيحاً في كلتا الحالتين.

إذا نُفذ السطر 5، وبناء على ما جاء في الشرط في السطر 3، يكون لدينا  $x.left = T.nil$  أو  $x.left.max < i.low$ . فإذا كان  $x.left = T.nil$  فمن الواضح أن الشجرة الفرعية التي جذرها  $x.left$  لا تحتوي على أي مجال يتراكب مع  $i$ ، وهكذا فإن وضع  $x.right$  في  $x$  يحافظ على اللامتغير. لذا، سنفترض أن  $x.left \neq T.nil$  وأن  $x.left.max < i.low$ . وكما بينه الشكل 5.14 (أ) فإنه لدينا في كل مجال  $i'$  في الشجرة الفرعية اليسرى لـ  $x$

$$i'.high \leq x.left.max < i.low.$$

واعتماداً على خاصية التفرّع الثلاثي للمحالات، فإن  $i'$  و  $i$  لا يتراكبان، ومن ثم، فإن الشجرة الفرعية اليسرى لـ  $x$  لا تحتوي على أي مجال يتراكب مع  $i$ ، وهكذا فإن وضع  $x.right$  في  $x$  يحافظ على اللامتغير.

من جهة أخرى، إذا نُفذ السطر 4، فإننا سنرى أن المكافئ العكسي لامتغير الحلقة يتحقق. بمعنى أنه إذا لم توجد مجالات تتراكب مع  $i$  في الشجرة الفرعية التي جذرها  $x.left$ ، فليس هناك مجال في أي مكان في الشجرة يتراكب مع  $i$ . وحيث إن السطر 4 قد نُفذ، بسبب تحقق الشرط في السطر 3، فإن  $x.left.max \geq i.low$ . وإضافة إلى ذلك، وبسبب تعريف الواصفة  $max$ ، يجب أن يكون هناك

بمجال ما  $i'$  في الشجرة الفرعية اليسرى ل  $x$  بحيث

$$\begin{aligned} l'.high &= x.left.max \\ &\geq i.low. \end{aligned}$$

(يوضح الشكل 5.14 (ب) هذه الحالة). وإذا إن  $i$  و  $i'$  لا يتراكبان، و  $i'.high < i.low$  غير محقق، فهذا يستتبع، بحسب خاصية التفريع الثلاثي للمحالات، أن  $i'.high < i.low$ . ونحمل عقد أشجار المجال مفتاح  $i$  توافق النقاط الطرفية الدنيا للمحالات، ومن ثم، نفتضي خاصية شجرة البحث أنه مهما يكن المجال  $i''$  في الشجرة الفرعية اليمنى ل  $x$ ، يكون لدينا

$$\begin{aligned} i.high &< i'.low \\ &\leq i''.low. \end{aligned}$$

حسب خاصية التفريع الثلاثي للمحالات، لا يتراكب  $i$  و  $i''$ . ونستنتج أنه سواء وُجد مجال في الشجرة الفرعية اليسرى ل  $x$  يتراكب مع  $i$  أم لم يوجد، فإن وضع  $x.left$  في  $x$  يحافظ على الامتصاف.

الانتهاء: عندما تنتهي الحلقة في حالة  $x = T.nil$ ، فهذا يعني أنه لا يوجد مجال يتراكب مع  $i$  في الشجرة الفرعية التي جذرها  $x$ . يقتضي المكافئ العكسي للامتصاف الحلقة أن  $T$  لا تحتوي على أي مجال يتراكب مع  $i$ . ومن ثم فإن إعادة  $x = T.nil$  هي نتيجة سليمة. ■

وهكذا، فإن الإجراء INTERVAL-SEARCH يعمل بطريقة سليمة.

تقارن

1-3.14

اكتب شبه الرمز ل LEFT-ROTATE الذي يعمل على العقد في شجرة مجالات ويُحدث المواصفات  $max$  في زمن  $O(1)$ .

2-3.14

أعد كتابة رماز INTERVAL-SEARCH بحيث يعمل بطريقة صحيحة عندما تكون جميع المجالات مفتوحة.

3-3.14

صف خوارزمية فعالة تعيد، في حالة وجود مجال  $i$  معطى، مجالاً يتراكب مع  $i$  بحيث تكون لديه أصغر نقطة طرفية دنيا، أو تعيد  $T.nil$  في حال عدم وجود مثل هذا المجال.

4-3.14

لنكن  $T$  شجرة مجالات معطاة، والمجال  $i$ . صف كيف يمكن سرد جميع المجالات في  $T$  التي تتراكب مع  $i$  خلال زمن  $O(\min(n, k \lg n))$ ، حيث  $k$  عدد المجالات في لائحة الخرج. (نلميح: توجد طريقة بسيطة تولّد عدة استعلامات، وتمثّل الشجرة بين الاستعلامات. ويوجد حل آخر أكثر تعقيداً بقليل لا يعدّل الشجرة.)



## 5-3.14

اقترح تعديلات على إجراءات شجرة الحالات بحيث تدعم العملية الجديدة INTERVAL-SEARCH-EXACTLY( $T, i$ )، حيث  $T$  شجرة بحالات، و  $i$  مجال. تعيد العملية مؤشرًا إلى عقدة  $x$  في  $T$  عندما يكون  $x.int.low = i.low$  و  $x.int.high = i.high$ ، أو تعيد  $T.nil$  إذا كانت  $T$  لا تحوي مثل هذه العقدة. يجب أن تتخذ جميع العمليات، ومن ضمنها INTERVAL-SEARCH-EXACTLY في زمن  $O(\lg n)$  في شجرة بحالات ذات  $n$  عقدة.

## 6-3.14

بين كيف نحافظ على مجموعة ديناميكية  $Q$  من الأعداد التي تدعم العملية MIN-GAP، وهي العملية التي تعطي طولاً الفرق بين أقرب عددين في  $Q$ . مثلاً إذا كانت  $Q = \{1, 5, 9, 15, 18, 22\}$  عندها تعيد MIN-GAP( $Q$ ) القيمة  $3 = 18 - 15$  لأن 15 و 18 هما أقرب عددين أحدهما من الآخر في  $Q$ . اجعل العمليات INSERT و DELETE و SEARCH و MIN-GAP فعالة قدر المستطاع وحلّ أزمنة تنفيذها.

## \* 7-3.14

تمثل قواعد معطيات VLSI عموماً دائرة متكاملة على شكل لائحة من المستطيلات. افترض أن كل مستطيل موشة بحيث نوازي أضلاعه المحويين  $x$  و  $y$ ، وبحيث تمثل كل مستطيل بأدنى وأعلى قيمة لإحداثياته على  $x$  و  $y$ . أعط خوارزمية تعمل في زمن  $O(n \lg n)$  لتحديد كون مجموعة المستطيلات  $n$  الممثلة بما ذكرناه أنفاً تحتوي على مستطيلات متراكبة، أو لا. لا تحتاج خوارزمتك إلى إيراد جميع الأزواج المتقاطعة، ولكنها يجب أن تشير إلى وجود تراكب في حالة غطى مستطيل آخر تماماً، وإن كانت خطوط المحيط لا تتقاطع. (لمصيح: حرك خطأ "ماسحاً" عبر مجموعة المستطيلات.)

## مسائل

## 1.14 نقطة التراكب الأعظمي

افترض أننا نرغب في تتبع نقطة التراكب الأعظمي *point of maximum overlap* في مجموعة من الحالات، وهي النقطة التي يتراكب فيها أكبر عدد من مجالات المجموعة.

أ. بين أنه يوجد دوماً نقطة تراكب أعظمي هي نقطة طرفية لإحدى المقتطعات.

ب. صمّم بنية معطيات تدعم دعماً فعالاً للعمليات INTERVAL-DELETE و INTERVAL-INSERT و FIND-POM التي تعيد نقطة التراكب الأعظمي. (لمصيح: احتفظ بشجرة حمراء-سوداء لجميع النقاط الطرفية، وأستد القيمة +1 لكل نقطة طرفية يسرى، والقيمة -1 لكل نقطة طرفية يمخى. أغن كل عقدة في الشجرة للمعلومات الإضافية اللازمة للمحافظة على نقطة التراكب الأعظمي.)

## 2.14 تبديل جوزفوس

تعرف مسألة جوزفوس *Josephus problem* كما يلي: لنفترض أن لدينا  $n$  شخصاً مصطفين دائرياً، ولدينا عدد صحيح موجب  $m \leq n$ . نختار أحد الأشخاص نقطة بداية، ونبدأ بالعدّ حول الدائرة بحيث نستبعد الشخص ذا الترتيب  $m$ . وبعد استبعاد الشخص، يستمر العدّ حول الدائرة الجديدة الناتجة. يستمر هذا الإجراء إلى أن يُستبعد جميع الأشخاص وعددهم  $n$ . إن الترتيب الذي جرى وفقه استبعاد الأشخاص من الدائرة يعرف بتبديل جوزفوس  $(n, m)$ -*Josephus permutation* على الأعداد  $1, 2, \dots, n$ . فمثلاً تبديل جوزفوس  $(7, 3)$  هو  $(3, 6, 2, 7, 5, 1, 4)$ .

أ. افترض أن  $m$  ثابت، وأن  $n$  عدد صحيح معطى. صِفْ خوارزمية زمنها  $O(n)$  تعطي خرجاً هو تبديل جوزفوس  $(n, m)$ .

ب. افترض أن  $m$  ليس ثابتاً، وأن  $n$  و  $m$  عددان صحيحان. صِفْ خوارزمية زمنها  $O(n \lg n)$  تعطي خرجاً هو تبديل جوزفوس  $(n, m)$ .

## ملاحظات الفصل

يصف Preparata و Shamos [282] في كتابهما عددًا من أشجار المجالات التي تظهر في الأدبيات المرجعية، مستشهدين بأعمال H. Edelsbrunner (1980) و E. M. McCreight (1981). يعرض الكتاب شجرة مجالات تتيح لنا، إذا أُعطينا قاعدة معطيات ساكنة ذات  $n$  مجالاً، أن نعدّد المجالات  $k$  التي تتراكب مع استعلام معطى خلال زمن  $O(k + \lg n)$ .



## تمهيد

يدرس هذا الباب ثلاث تقنيات تُستعمل في تصميم الخوارزميات الفعالة وتحليلها: البرمجة الديناميكية (الفصل 15)، والخوارزميات الشرهة (الفصل 16)، والتحليل للمخشد (الفصل 17). وقد غرّضت الأجزاء السابقة تقنيات أخرى واسعة التطبيق، مثل تقنية فُرق-تشد *divide and conquer*، وتقنيات إضافة العشوائية وكيفية حل التكرارات. على أن التقنيات الواردة في هذا الباب أكثر تطوراً وتعقيداً إلى حد ما، غير أنها مفيدة لنا في معالجة الكثير من المسائل الحسابية، علماً بأن الأفكار الأساسية لموضوعات هذا الباب ستكرر لاحقاً في الكتاب.

تُطبق البرمجة الديناميكية نموذجاً في مسائل الأمثلة التي تتطلب أخذ مجموعة من الخيارات للوصول إلى الحل الأمثل. ومع كل خيار يُتخذ، كثيراً ما تنشأ مسائل جزئية (ثانوية) لها طابع المسائل الأصلية نفسه. وتكون البرمجة الديناميكية فعالة حين يمكن أن تنشأ مسألة جزئية من أكثر من مجموعة جزئية واحدة من الخيارات؛ وتمثل التقنية الرئيسية في عزز حلول جميع المسائل الجزئية هذه، فلرما عادت إلى الظهور ثانية. ويبدأ الفصل 15 كيف تستطیع هذه الفكرة السهلة، أحياناً، أن تحوّل خوارزميات ذات زمن أسّي إلى خوارزميات ذات زمن كثير حدودي.

تُطبق الخوارزميات الشرهة، شأن خوارزميات البرمجة الديناميكية على مسائل الأمثلة التي تتطلب تحديد مجموعة من الخيارات للوصول إلى الحل الأمثل. تكمن فكرة الخوارزمية الشرهة في تحديد كل خيار بطريقة أمثلة محلياً. ومن الأمثلة البسيطة على هذا "صرف القطع النقدية": فليكني بجعل عدد القطع النقدية الأمريكية اللازمة لصرف مبلغ معيّن من المال أصغر؛ يكفي أن نكرر اختيار أكبر فئة قطعة نقدية بحيث لا تتجاوز المبلغ المتبقي. ويتيح النهج الشره حلاً أمثلاً لمسائل كثيرة كهذه بسرعة أكبر بكثير مما قد يتيحها نهج البرمجة الديناميكية. ومع ذلك، فليس من السهل دوماً الحكم على أن النهج الشره سيكون فعالاً أم لا. ويعرف الفصل 16 نظرية الكيانات المصفوفية *matroid theory*، التي توفر أساساً رياضياً يساعدنا على إثبات أن الخوارزمية الشرهة تعطي حلاً أمثلاً.

ونستخدم التحليل المخطط لتحليل خوارزميات معينة تنجز متتالية من العمليات المتشابهة. وعوضاً عن المخطط من كلفة متتالية من العمليات بالمخطط من الكلفة الفعلية لكل عملية منفردة، يمكن استخدام التحليل المخطط لتزويدنا بالمخطط الأعلى للكلفة الفعلية لكامل المتتالية. ومن مزايا هذا النهج أنه في حين قد تكون بعض العمليات مكلفة، فثمة الكثير من العمليات الأخرى التي قد تكون رخيصة. بكلمات أخرى، يمكن أن تُنفَّذ العديد من العمليات في زمن أقل بكثير من زمن التنفيذ في أسوأ الحالات. على أن التحليل المخطط ليس مجرد أداة تحليل، ولكنه أيضاً طريقة للتفكير في تصميم الخوارزميات، بالنظر إلى أن تصميم خوارزمية وتحليل زمن تنفيذها عمليتان مترابطتان غالباً إلى حد بعيد. يقدم الفصل 17 ثلاث طرق لتحفيز التحليل المخطط لخوارزمية ما.

البرمجة الديناميكية، شأن طريقة "فرق-تشد"، تحلُّ مسائلَ بعضِ حلولِ مسائل جزئية. (تشير "البرمجة" في هذا السياق إلى طريقةٍ مُجدولة، وليس إلى كتابة رماز حاسوبي.) وكما رأينا في الفصلين 2 و 4، فإن خوارزميات فرق-تشد تجزئ المسألة إلى مسائل جزئية منفصلة (مستقلة)، وتُحلُّ للمسائل الجزئية عودياً، ثم تَضمُّ حلولها بغية حلِّ المسألة الأصلية. بالمقابل، تُطبِّق البرمجة الديناميكية dynamic programming حين لا تكون المسائل الجزئية مستقلة، أي حين تشارك للمسائل الجزئية subproblems في مسائل أكثر جزئية subsubproblems. وفي هذا السياق، فإن خوارزمية "فرق-تشد" تقوم بعمل أكثر من المطلوب، وذلك بحل المسائل الأكثر جزئية تكرارياً (مرة بعد مرة). أما خوارزمية البرمجة الديناميكية فتحلُّ كلَّ مسألة جزئية (أكثر جزئية) مرة واحدة فقط ثم تحتفظ بالإجابات في جدول، متلافيةً بذلك العمل على إعادة حساب الجواب في كل مرة تحلُّ فيها كلُّ مسألة جزئية ثانوية.

تُطبِّق البرمجة الديناميكية نموذجاً على مسائل الأمثلة optimization problems. ومثل هذه المسائل قد تنطوي على عدة حلول ممكنة، لكلِّ حلٍّ منها قيمة، ونودُّ إيجاد الحل ذي القيمة المثلى (الصغرى أو العظمى). نسمي مثل هذا الحل حلاً أمثل an optimal solution للمسألة، مقابل ما يسمى الحل الأمثل the optimal solution، إذ من الممكن أن توجد عدة حلول تحقِّق هذه القيمة المثلى.

ولدى تطوير خوارزمية برمجة ديناميكية، نتبع متاليةً من أربع خطوات:

1. وصف البنيان لحلِّ أمثل.
2. تعريف تكرارياً القيمة لحلِّ أمثل.
3. احسب القيمة لحلِّ أمثل بطريقة صعودية من القعر إلى القمة.
4. أنشئ حلاً أمثل مستقى من المعلومات المحسوبة.

تؤلّف الخطوات 1-3 الأساس لحلِّ مسألة بالبرمجة الديناميكية. وإذا اقتصرنا حاجتنا على قيمة حلِّ أمثل، لا نلزم الحل الأمثل نفسه، فيمكننا عندئذٍ حذف الخطوة الرابعة. وعند إنجازنا الخطوة الرابعة، تحتفظ أحياناً

معلومات إضافية أثناء حساب الخطوة الثالثة لتسهيل إنشاء حل أمثل.

في المقاطع الآتية نستخدم طريقة البرمجة الديناميكية لحل بعض مسائل الأمثلة؛ فدرس المقطع 1.15 مسألة تقطيع قضيب إلى قضبان أقصر طولاً بحيث نجعل قيمتها الكلية عظمى. ويطرح للمقطع 2.15 كيف يمكن ضرب سلسلة من المصفوفات بينما ننجز أقل عدد كلي من الجداءات السلمية. وفي ضوء هذه الأمثلة على البرمجة الديناميكية، يناقش للمقطع 3.15 خاصيتين أساسيتين يجب أن تتمتع بهما مسألة ما، كي تكون تقنية حلها بالبرمجة الديناميكية قابلة للتطبيق. ثم يبين للمقطع 4.15 طريقة إيجاد أطول متتالية جزئية مشتركة لمتتاليتين. أخيراً نستخدم المقطع 5.15 البرمجة الديناميكية لبناء أشجار بحث ثنائية أمثلة، إذا عُلمَ تنوع المفاتيح المنشودة.

## 1.15 تقطيع القضبان

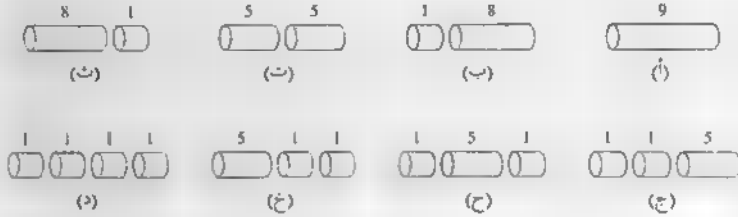
يستخدم مثالاً الأول البرمجة الديناميكية لحل مسألة بسيطة تتعلق بتحديد مكان قطع قضبان فولاذية. تشتري شركة سيرلينك Serling Enterprises قضباناً فولاذية طويلة وتقصعها إلى قضبان أقصر، ثم تبيعها. تجري عمليات القطع بدون تكلفة. وتود إدارة شركة Serling معرفة أفضل طريقة لقطع القضبان. نفترض أننا نعلم القيمة  $p_i$  بالدولار، حيث  $i = 1, 2, \dots$ ، التي تطلبها شركة Serling عن كل قضيب طوله  $i$  إنشاء، علماً بأن أطوال القضبان هي أعداد صحيحة من الإنشات دوماً. يعطي الشكل 1.15 مثالاً على جدول الأسعار.

إن مسألة تقطيع القضبان *rod-cutting problem* هي التالية: لدينا قضيب طوله  $n$  إنشا وحدول  $p_i$  بالأسعار، حيث  $i = 1, 2, \dots, n$ . حدد الدخل الأعظم  $r_n$  الذي يمكن الحصول عليه من تقطيع القضيب وبيع القطع. لاحظ أنه إذا كان السعر  $p_n$  لقضيب طوله  $n$  كبيراً بقدر كافٍ، فقد لا يتطلب الحل الأمثل أي عملية قطع على الإطلاق.

لندرس الحالة التي فيها  $n = 4$ . يبين الشكل 2.15 كل الطرق لقطع قضيب طوله 4 إنشات، ومنها الطريقة التي ليس فيها أية عملية قطع على الإطلاق. نرى أن تقطيع القضيب إلى قطعتين، طول كل منهما إنشان، يعطي دخلاً  $10 = 5 + 5 = p_2 + p_2$ ، وهو دخل أمثل.

الطول $i$	1	2	3	4	5	6	7	8	9	10
السعر $p_i$	1	5	8	9	10	17	17	20	24	30

الشكل 1.15 مثال على جدول أسعار القضبان. كل قضيب طوله  $i$  إنشا يُكسب الشركة دخلاً قدره  $p_i$  دولاراً.



**الشكل 2.15.** الطرق الثمانية الممكنة لتقطيع قضيب طوله 4. يشر الرقم فوق كل قطعة إلى قيمة تلك القطعة، بنسب جدول مثال الأعداد المبين في الشكل 1.15. الاستراتيجية لكلي هي الجزء (ت)، تقطيع القضيب إلى قطعتين طول كل منهما 2، والتي قيمتها الكلية هي 10.

يمكننا تقطيع قضيب طوله  $n$  بـ  $2^{n-1}$  طريقة مختلفة، إذ لدينا خيار مستقل للقطع أو عدم القطع، عند المسافة  $i$  إنشأنا من الطرف الأيسر، حيث  $i = 1, 2, \dots, n-1$ .<sup>1</sup> سنشير إلى التجزئة إلى قطع باستخدام تدوين الجمع العادي، بحيث تشير العلاقة  $7 = 2 + 2 + 3$  إلى تقطيع قضيب طوله 7 لإنشآت إلى ثلاث قطع؛ اثنتان بطول 2 والثالثة بطول 3. إذا قطع حل أمثل القضيب إلى  $k$  قطعة، حيث  $k \leq n$ ، عندها نُقدِّم تجزئة مثلى للقضيب

$$n = i_1 + i_2 + \dots + i_k$$

إلى قطع أطوالها  $i_1, i_2, \dots, i_k$  دخلياً أعظمياً موافقاً قدره:

$$r_n = p_{i_1} + p_{i_2} + \dots + p_{i_k}.$$

يمكننا بالاستقصاء، في مثال مسألتنا، تحديد العوائد المثلى  $r_i$  في حال  $i = 1, 2, \dots, 10$ ، مع التجزئة

المثلى الموافقة

$$r_1 = 1 \quad \text{من الحل } 1 = 1 \text{ (لا يوجد أي قطع)،}$$

$$r_2 = 5 \quad \text{من الحل } 2 = 2 \text{ (لا يوجد أي قطع)،}$$

$$r_3 = 11 \quad \text{من الحل } 3 = 3 \text{ (لا يوجد أي قطع)،}$$

<sup>1</sup> إذا أردنا أن تكون الأجزاء المقطعة بترتيب القياس غير المتناقص، كان لدينا عدد أقل من طرق التقطيع نؤخذ بالحسبان. ففي حالة  $n = 4$  يكون طرق فقط كهذه: الأجزاء (أ) و (ب) و (ت) و (ث) و (ج) في الشكل 2.15. يسمى عدد الطرق دالة التجزئة partition function؛ وهو يساوي تقريباً  $e^{\pi\sqrt{2n/3}}/4n\sqrt{3}$ . وهذه الكمية أقل من  $2^{n-1}$ ، ولكنها تبقى أكبر بكثير من أي كثير حدود في  $n$ . إلا أننا لن نتابع هذا الخط من النقاش لاحقاً.



$$r_1 = \text{من حل } 0 + 0 = 0$$

$$r_2 = \text{من حل } 0 + 1 = 1$$

$$r_3 = \text{من حل } 0 + 0 = 0 \text{ (لا يوجد أي قطع)}$$

$$r_4 = \text{من حل } 7 + 0 = 7$$

$$r_5 = \text{من حل } 8 + 2 = 10$$

$$r_6 = \text{من حل } 9 + 3 = 12$$

$$r_7 = \text{من الحل } 10 = 10 \text{ (لا يوجد أي قطع)}$$

وتعصب أكثر، يمكننا استنباط القيمة  $r_n$  في حال  $n \geq 1$  بدلالة الدخول مسبق من نفس المسألة:

$$r_n = \max_{1 \leq i \leq n} (r_i + r_{n-i}) \quad (1.15)$$

يوفق تحديد الأول  $p_n$  عدة إجراء في قطع على الإطلاق، ويصنع تقطيع ذاتي عليه  $p_n$  كما هو. ثم حددت أن  $n-1$  الأخرى تتوافق لدخل الأعظم الذي نحصل عليه بإجراء قطع ذاتي تقطيع  $p_{n-1}$  أو تقطين مرفوضاً  $i$  و  $n-i$ ، لكل قيم  $i = 1, 2, \dots, n-1$ . ثم تقطيع هذه القطع تقطيع ذاتي آخر. نحصل على لدخول  $r_i$  و  $r_{n-i}$  من هاتين التقطعتين. ولأننا لا نعلم مقدماً أي قيمة  $i$  نجعل لدخول أعظم، عند دراسة جميع القيم الممكنة  $i$  وأخذ قيمة  $i$  نجعل لدخول أعظم. لدينا أيضاً خيار عدة أخذ أي قيمة  $i$  إذا استطعنا الحصول على دخل كما يصح تقطيع دون قطع.

لاحظ أنه لحل المسألة الأصلية في حجمنا  $n$ ، نحل مسائل أصغر من نوع غش ويمكن بحجوه أصغر، وبإجراء القطع الأول. نجد عدة فصول متسلسلة مستقلة من مسألة تقطيع تقطعتين. يضم الحل الأمثل الكلي حلول مسائل متسلسلة متسلسلة متسلسلة بتعظيم الدخل من كل من تقطعتين. نقول إن مسألة تقطيع المقصات هي مسألة *optimal substructure*: أي إن حلولاً مثلى لمسألة تتضمن حلولاً مثلى لمسائل أصغر حجم من نوع غش هي حلول مثلى للمسألة نفسها.

نصف المسألة الأولى هي مسألة تقطيع الفطيان، لكنها أبسط قليلاً، فإننا ننظر إلى التداخل من حل المسألة الأولى. ونلاحظ أن الطرف الأيسر، ثم بقية من الطرف اليمين متطابقان. المسألة الأولى هي المسألة الأولى، والتي يمكن تجزئتها أكثر فأكثر. ونلاحظ أن المسألة الأولى هي المسألة الأولى. كقطعة أولى متنوعة ببعض التجزئات للقطعة نفسها. ونلاحظ أن المسألة الأولى هي المسألة الأولى. نلاحظ أن طول القطعة الأولى هو  $p_n$  (الدخل الموافق لها  $r_n = 0$ ). وبذلك نحصل على النسخة

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i}) \quad (1.16)$$

في هذه الصياغة، يتضمن الحل الأمثل حلاً لمسألة جزئية مرتبطة واحدة فقط - وهي القطعة المتبقية - عوضاً عن مسألتين جزئيتين.

### تنفيذ نزولي عودي

نفذ الإجراء التالي الحسابات الضمنية في المعادلة (2.15) بطريقة مباشرة نزولية top-down عودية.

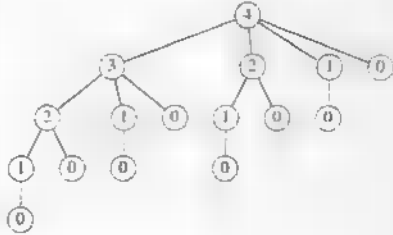
```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

ياخذ الإجراء CUT-ROD كمدخل صيغة  $p[1..n]$  للأسعار وعددًا صحيحًا  $n$ ، ويعيد الدخل الأعظم الممكن لقطيب طوله  $n$ . فإذا كانت  $n = 0$ ، فلا يوجد أي دخل ممكن، وبذلك يعيد الإجراء CUT-ROD القيمة 0 في السطر 2. يستدئ السطر 3 بحمل قيمة الدخل الأعظمي  $q$  تساوي  $-\infty$ ، بحيث تحسب الحلقة for في السطرين 4 و 5 القيمة الصحيحة لـ  $q = \max_{1 \leq i \leq n} (p[i] + \text{CUT-ROD}(p, n - i))$  ثم يعيد السطر 6 هذه القيمة. وبافتراض بسيط لـ  $n$  يتبين أن هذا الجواب يساوي فعلاً الجواب المرغوب فيه  $r_n$  باستخدام المعادلة (2.15).

وإذا كان عليك كتابة رماز الإجراء CUT-ROD باستخدام لغة البرمجة المفضلة لديك وتنفيذه على حاسوبك، لوجدت أنه حلماً يصبح طول الدخل كبيراً كبيراً متوسطاً، يستغرق تنفيذ برنامجك وقتاً طويلاً. فحين تكون  $n = 40$ ، ستجد أن برنامجك يستغرق عدة دقائق على الأقل، وربما استغرق أكثر من ساعة. وستجد عملياً أن زمن تنفيذ برنامجك يتضاعف تقريباً كلما زادت قيمة  $n$  بمقدار 1.

ما هو سبب عدم فعالية CUT-ROD؟ تكمن للمشكلة في أن CUT-ROD تستدعي نفسها عودياً مرات ومرات بنفس قيم للتوسطات؛ فهي إذن تحل للمسائل الجزئية ذاتها تكرارياً. يبين الشكل 3.15 ما يحدث في حالة  $n = 4$ :  $\text{CUT-ROD}(p, n)$  تستدعي  $\text{CUT-ROD}(p, n - i)$  لقيم  $i = 1, 2, \dots, n$ . وهذا يكافئ استدعاء  $\text{CUT-ROD}(p, n)$  لـ  $\text{CUT-ROD}(p, j)$  لقيم  $j = 0, 1, \dots, n - 1$ . وحين نفلطط إلى الإجراءات عودياً، فإن حجم العمل يزداد ازدياداً انفجارياً بدلالة  $n$ .

ولتحليل زمن تنفيذ CUT-ROD، لنشر بـ  $T(n)$  إلى عدد الاستدعاءات الكلي للإجراءات CUT-ROD حين يكون المتوسط الثاني له مساوياً  $n$  عند استدعائه. وهذا التعبير يساوي عدد العقد في الشجرة الفرعية التي لصيقة جذورها  $n$  في الشجرة العودية. يتضمن العد الاستدعاء الأولى عند الجذر. وبذلك



**الشكل 3.15** الشجرة العودية التي تبين الاستدعاءات العودية الناتجة عن استدعاء  $CUT-ROD(p, n)$  في حالة  $n = 4$ . تعطي كل لصيقة عقدة الحجم  $n$  للمسألة الجزئية الموافقة، وبذلك توافق الوصلة من أب نصيفته  $s$  إلى ابني نصيفته  $t$  تقطيع قطعة أولية حجمها  $s - t$  وترك مسائل جزئية متبقية حجمها  $t$ . اشارة من الجذر إلى ورقة توافق إحدى الطرق الـ  $2^{n-1}$  لقطع قضيب طوله  $n$ . وعموماً، يكون للشجرة العودية هذه  $2^n$  عقدة و  $2^{n-1}$  ورقة.

يكون،  $T(0) = 1$  و

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) . \quad (3.15)$$

الرقم 1 هو للاستدعاء عند الجذر، والحد  $T(j)$  يحصى عدد الاستدعاءات (ومنها الاستدعاءات العودية) الناتجة عن الاستدعاء  $CUT-ROD(p, n - i)$ ، حيث  $i = n - j$ . يُطلب إليك في التمرين 1-1.15 أن تبين أن:

$$T(n) = 2^n , \quad (4.15)$$

وبذلك يكون زمن تنفيذ  $CUT-ROD$  أسياً في  $n$ .

وبإعادة النظر فيما قد سلف نجد أن زمن التنفيذ الأسّي هذا ليس مفاجئاً كثيراً؛ فمن الواضح أن  $CUT-ROD$  تأخذ بالحسبان كل الطرق  $2^{n-1}$  الممكنة لقطع قضيب طوله  $n$ . ويُذكر أن لشجرة الاستدعاءات العودية  $2^{n-1}$  ورقة، واحدة لكل طريقة ممكنة لقطع القضيب. تعطي اللصقات على المسار البسيط من الجذر إلى إحدى الأوراق حجوماً كل القطع المتبقية اليمنى قبل كل عملية قطع؛ أي إن اللصقات تعطي نقاط القطع الموافقة، فقيسة من الطرف الأيمن للقضيب.

### استخدام البرمجة الديناميكية لقطع القضبان الأمثل

سنبين الآن كيف نحول  $CUT-ROD$  إلى خوارزمية فعالة باستخدام البرمجة الديناميكية.

تعمل طريقة البرمجة الديناميكية كما يلي. بعد ملاحظة عدم فعالية الحل العودي البسيط لأنه يحل المسائل الجزئية نفسها تكرارياً، نتخذ الترتيبات اللازمة لحل كل مسألة جزئية مرة واحدة فقط، وتخزين حلّها. فإذا أردنا العودة إلى حل هذه المسألة الجزئية ثانية لاحقاً، يكفي أن نتفقدنا دون الحاجة إلى إعادة حسابها. وبذلك

تستخدم البرمجة الديناميكية ذاكرة إضافية لاختصار زمن الحساب؛ وهي بذلك تقدم مثالاً على **المقايضة بين الزمن والذاكرة** *time-memory trade-off*. ويمكن أن يكون الكسب الزمني هائلاً؛ إذ يمكن تحويل الحل بزمنٍ أسي إلى حلٍ بزمنٍ كثير حدودي. ويُقَدِّم أسلوب البرمجة الديناميكية بزمن كثير حدودي عندما يكون عدد المسائل الجزئية المتمايزة ذات الصلة كثير حدودي في حجم المدخل، ويمكننا حل كل مسألة جزئية منها بزمن كثير حدودي.

يوجد عادةً طريقتان متكافئتان لننجز أسلوب البرمجة الديناميكية، سنوضحهما في مثالنا المتعلق بقطع القضبان.

الأسلوب الأول **نزولي مع استلكار** *top-down with memoization*<sup>2</sup>، وبمقتضاه نكتب الإجراء عودياً بطريقة طبيعية، ولكن نُعدِّله بحيث نخزن نتيجة كل مسألة جزئية (عادةً في صيغة أو جدول تليبد). الإجراء الآن يتحقق أولاً ليرى إن كانت المسألة الجزئية محلولة سابقاً. فإذا كان الأمر كذلك، أعاد القيمة المخزنة، مختصراً مهزداً من الحسابات عند هذا المستوى؛ وإلا، فإنه يحسب القيمة بالطريقة المعتادة. ونقول إن الإجراء العودي قد جرى **استلكاره** *memoized*؛ فهو "يتذكر" النتائج التي حسبها سابقاً.

أما الأسلوب الثاني فهو **الطريقة الصعودية** *bottom-up method*. يعتمد هذا الأسلوب عادةً على مفهوم طبيعي ما لـ "حجم" المسألة الجزئية، بحيث يعتمد حلُّ مسألة جزئية معينة اعتماداً كاملاً على حلِّ مسائل جزئية "أصغر". نغرز المسائل الجزئية وفقاً لحجمها ونحلها بحسب ترتيب حجمها بحيث نبدأ بأصغرها. فإذا أتينا إلى حلِّ مسألة جزئية معينة، نكون قد حللنا قبلها كل المسائل الجزئية التي هي أصغر منها، والتي يعتمد عليها حل هذه المسألة الجزئية، وعزّينا تلك الحلول. نحلُّ كلَّ مسألة جزئية مرةً واحدة فقط، وعندما نراها أول مرة نكون قد حللنا كل المسائل الجزئية التي تتطلبها.

هذان الأسلوبان يعطيان خوارزميتين لهما زمن التنفيذ المقارب نفسه، باستثناء الظروف الاستثنائية؛ حيث يتمتع الأسلوب النزولي عن العمل عودياً لسر جميع المسائل الجزئية الممكنة. أما الأسلوب الصعودي فيتمتع غالباً بعوامل ثابتة أفضل، لأن حملها الإضافي من استدعاءات الإجراء أقل.

نورد فيما يلي شبه الرمز للإجراء النزولي CUT-ROD مع إضافة الاستدكار:

MEMOIZED-CUT-ROD( $p, n$ )

- 1 let  $r[0 \dots n]$  be a new array
- 2 for  $i = 0$  to  $n$
- 3      $r[i] = -\infty$
- 4 return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

<sup>2</sup> ليس هذا خطأ في رسم الكلمة؛ فهي *memoization* فعلاً، وليس *memorization*. وهي مستفدة من *memo*، بالنظر إلى أن التقنية تتمثل في تسجيل قيمةٍ يمكننا العودة إليها لاحقاً.

MEMOIZED-CUT-ROD-AUX( $p, n, r$ )

```

1  if  $r[n] \geq 0$ 
2      return  $r[n]$ 
3  if  $n == 0$ 
4       $q = 0$ 
5  else  $q = -\infty$ 
6      for  $i = 1$  to  $n$ 
7           $q = \max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r))$ 
8   $r[n] = q$ 
9  return  $q$ 

```

هنا، يستبدل الإجراء الرئيسي MEMOIZED-CUT-ROD صفيحة مساعدة جديدة  $r[0..n]$  قيمها تساوي  $-\infty$ ، وهو خيار مناسب نشير به إلى "مجهول". (قيم الدخل revenue المعروفة غير سالبة دوماً). ثم يستدعي مسافة المساعد MEMOIZED-CUT-ROD-AUX.

الإجراء MEMOIZED-CUT-ROD-AUX هو النسخة مع الاستدعاء لإجراءنا السابق CUT-ROD. إنه يتفق أولاً في السطر 1 إذا كانت القيمة المرغوبة معروفة سابقاً، فإذا كان الأمر كذلك، يعيد هذه القيمة في السطر 2. وإلا فإنه يحسب في الأسطر 3-7 القيمة المرغوبة  $q$  بالترقبة المتعددة، السطر 8 يحزن القيمة في  $r[n]$ ، والسطر 9 يعيدها. النسخة الصغودية فهي أبسط:

BOTTOM-UP-CUT-ROD( $p, n$ )

```

1  let  $r[0..n]$  be a new array
2   $r[0] = 0$ 
3  for  $j = 1$  to  $n$ 
4       $q = -\infty$ 
5      for  $i = 1$  to  $j$ 
6           $q = \max(q, p[i] + r[j - i])$ 
7       $r[j] = q$ 
8  return  $r[n]$ 

```

في حالة الأسلوب الصغودي للبرمجة الديناميكية، يُستخدم الإجراء BOTTOM-UP-CUT-ROD الترتيب الطبيعي للمسائل الجزئية: فمسألة جزئية حجمها  $i$  "أصغر" من مسألة جزئية حجمها  $j$  إذا كان  $i < j$ . وبذلك يحل الإجراء المسائل الجزئية التي حجمها  $n, n-1, \dots, 0$ ، في هذا الترتيب.

السطر 1 في الإجراء BOTTOM-UP-CUT-ROD ينشئ صفيحة جديدة  $r[0..n]$  يحزن فيها نتائج المسائل الجزئية، والسطر 2 يستبدل  $r[0]$  بالقيمة 0 لأن القضيب الذي طوله 0 لا يكسب أي دخل. تحل السطور 3-6 كل مسألة جزئية حجمها  $j$  لكل  $j = 1, 2, \dots, n$  بترتيب الحجم للزيادة. والأسلوب الذي

استُخدم في حل مسألة بحجم معين  $n$  هو نفسه المُستخدَم في CUT-ROD، باستثناء أن السطر 6 يعنون الآن مباشرة عنصر الصيغة  $r[j-i]$  بدلاً من القيام باستدعاء عَوْدِي لحل المسألة الجزئية التي حجمها  $i-j$ . يُخزن السطر 7 في  $r[j]$  حلّ للمسألة الجزئية التي حجمها  $j$ . أخيراً، يعيد السطر 11 القيمة  $r[n]$  التي تساوي القيمة المثلى  $r_n$ .

للسنختين الصعودية والتّزولية زمن التنفيذ المقارب نفسه. زمن تنفيذ الإجراء BOTTOM-UP-CUT-ROD هو  $\Theta(n^2)$  بسبب بنية الحلقتين للتداخلتين. يشكل عدد التكرارات في حلقة for الداخلية، في السطور 5-6 سلسلة حسابية. زمن تنفيذ نظيره التّزولي MEMOIZED-CUT-ROD هو أيضاً  $\Theta(n^2)$ ، مع أن رؤية زمن التنفيذ هذا أصعب بقليل. ولما كان الاستدعاء العَوْدِي لحل مسألة جزئية جرى حلها سابقاً بعيد القيمة لوزا، فإن MEMOIZED-CUT-ROD يحلّ كلّ مسألة جزئية مرة واحدة فقط. إنه يحلّ المسائل الجزئية ذات الحجم  $0, 1, \dots, n$ . وحلّ مسألة جزئية حجمها  $n$ ، فإن حلقة for في السطرين 6-7 تكرر  $n$  مرة. وبذلك يشكل عدّد التكرارات الكلي لحلقة for هذه، لكل الاستدعاءات العودية لـ MEMOIZED-CUT-ROD سلسلة حسابية، تغطي تكرارات عددها  $\Theta(n^2)$ ، تماماً مثل حلقة for الداخلية في BOTTOM-UP-CUT-ROD. (إننا فعلياً نستخدم نوعاً من التحليل المُجمّع هنا، وسعالج التحليل المُجمّع لاحقاً في المقطع 1.17.)

#### بيانات المسائل الجزئية

حين نفكر في مسألة برمجة ديناميكية، علينا أن ندرك مجموعة المسائل الجزئية ذات الصلة، وكيف تعتمد هذه المسائل الجزئية بعضها على بعض.

يضم بيان المسألة الجزئية *subproblem graph* للمسألة هذه المعلومات تماماً. يوضّح الشكل 4.15 بيان للمسألة الجزئية لمسألة تقطيع القضبان في حالة  $n = 4$ . إنه يبيّن موجه، يتضمن عقدة واحدة لكل مسألة جزئية متميزة. وليبيان المسألة الجزئية وصلة موجهة من عقدة للمسألة الجزئية  $x$  إلى عقدة للمسألة الجزئية  $y$  إذا كان تحديد حل أمثل للمسألة الجزئية  $x$  يتطلب الأخذ بالحسبان حلاً أمثلاً للمسألة الجزئية  $y$ . على سبيل المثال، يتضمن بيان المسألة الجزئية وصلة من  $x$  إلى  $y$  إذا كان الإجراء العَوْدِي التّزولي لحلّ  $x$  يستدعي نفسه مباشرة لحلّ  $y$ . يمكننا النظر إلى بيان المسألة الجزئية على أنه نسخة "مختصرة" أو "مدبجة" للشجرة العودية في الطريقة العودية التّزولية، التي تدمج فيها كل المقادير المتعلقة بالمسألة الجزئية نفسها في عقدة واحدة، ونوجه كل الوصلات من الأب إلى الابن.

تُعتبر الطريقة الصعودية للبرمجة الديناميكية عُقد بيان المسألة الجزئية مرتبة بحيث نحلّ للمسائل الجزئية  $y$  المجاورة لمسألة جزئية معلومة  $x$  قبل حلّ للمسألة الجزئية  $x$ . (نذكر من القطع ب.4 أن علاقة التحاور ليست متناظرة بالضرورة.) وباستخدام مصطلحات الفصل 22، لخوارزمية البرمجة الديناميكية الصعودية، فإننا نعتبر العقدة في بيان للمسألة الجزئية مرتبة حسب "فرز طبولوجي معكوس" أو "فرز طبولوجي للمنتول"



**الشكل 4.15** بيان المسألة الجزئية لمسألة تقطيع القضبان في حالة  $n = 4$ . تعضي لصيقات العقد حجوم المسائل الجزئية الموافقة. تشير الوصلة الموجهة  $(x, y)$  إلى أننا نحتاج إلى حل المسألة الجزئية  $y$  حين نحل المسألة الجزئية  $x$ . بشكل هذا البيان نسخة مختصرة لشجرة البيان في الشكل 3.15، التي تندمج فيها جميع العقد التي لها اللصيقة نفسها في عقدة واحدة، وتطلق جميع الوصلات من الأب إلى الابن.

(انظر المقطع 4.22) لبيان المسألة الجزئية. وبعبارة أخرى، لا ندرس أي مسألة جزئية حتى نحل كل المسائل الجزئية التي تعتمد عليها. وبالمثل، وباستخدام مفاهيم من الفصل نفسه، يمكننا النظر إلى الطريقة التكرارية (مع استذكار) للبرمجة الديناميكية على أنها "بحث في العمق أولاً" لبيان المسألة الجزئية (انظر المقطع 3.22).

يمكن أن يساعدنا حجم بيان المسألة الجزئية  $G = (V, E)$  على تحديد زمن تنفيذ خوارزمية البرمجة الديناميكية. ولأننا نحل كل مسألة جزئية مرة واحدة فقط، فإن زمن التنفيذ هو مجموع الأزمنة اللازمة لحل كل مسألة جزئية. ويكون زمن حساب حل مسألة جزئية متناسباً عادةً مع درجة العقدة الموافقة في بيان المسألة الجزئية (عدد الوصلات الخارجة منها)، وعدد المسائل الجزئية مساوياً عدد العقد في بيان المسألة الجزئية. في هذه الحالة العامة، يكون زمن تنفيذ البرمجة الديناميكية خطياً من جهة عدد العقد والوصلات.

### إعادة بناء الحل

تعيد حلولنا لمسألة تقطيع القضبان باستخدام البرمجة الديناميكية قيمة الحل الأمثل، إلا أننا لا نعيد حلاً فعلياً: لائحة بأطوال القطع. يمكننا توسيع أسلوب البرمجة الديناميكية ليسجل ليس فقط القيمة المثلى المحسوبة لكل مسألة جزئية، وإنما أيضاً الخيار الذي أفضى إلى القيمة المثلى. بهذه المعلومات، يمكننا الآن طباعة حل أمثل.

وفيما يلي نسخة موسعة من BOTTOM-UP-CUT-ROD تحسب، لكل قضيب طوله  $i$ ، ليس فقط الدخل الأعظم  $r_i$ ، بل الطول الأمثل  $s_i$  أيضاً لأول قطعة يجب قطعها:

### EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )

```

1  let  $r[0..n]$  and  $s[0..n]$  be new arrays
2   $r[0] = 0$ 
3  for  $j = 1$  to  $n$ 
4       $q = -\infty$ 
5      for  $i = 1$  to  $j$ 
6          if  $q < p[i] + r[j - i]$ 
7               $q = p[i] + r[j - i]$ 
8               $s[j] = i$ 
9       $r[j] = q$ 
10 return  $r$  and  $s$ 

```

هذا الإجراء مشابه لـ BOTTOM-UP-CUT-ROD، باستثناء أنه ينشئ المصفوفة  $s$  في السطر 5، ويُحدَّث  $s[j]$  في السطر 8 ليحتفظ بالطول الأمثل  $i$  لأول قطعة نقطعها عند حل المسألة الجزئية التي حجمها  $j$ .

يأخذ الإجراء التالي لائحة الأسعار  $p$  وطول القضيب  $n$ ، ويستدعي EXTENDED-BOTTOM-UP-CUT-ROD لحساب المصفوفة  $s[1..n]$  التي تحتل أطوال القطع الأول لثلاثي، ثم يطبع اللائحة الكاملة بأطوال القطع في تقسيم أمثل لقضيب طوله  $n$ :

### PRINT-CUT-ROD-SOLUTION( $p, n$ )

```

1  ( $r, s$ ) = EXTENDED-BOTTOM-UP-CUT-ROD( $p, n$ )
2  while  $n > 0$ 
3      print  $s[n]$ 
4       $n = n - s[n]$ 

```

في مثالنا لقطع القضبان، فإن الاستدعاء EXTENDED-BOTTOM-UP-CUT-ROD( $p, 10$ ) سيعيد المصفوفتين التاليين:

	10	9	8	7	6	5	4	3	2	1	0	$i$
$r[i]$	30	25	22	■	17	13	10	■	5	1	0	
$s[i]$	10	3	2	1	6	2	2	3	2	1	0	

إن استدعاء PRINT-CUT-ROD-SOLUTION( $p, 10$ ) سيطبع 10 فقط، إلا أن استدعاءها مع  $n = 7$  سيطبع القطعتين 1 و 6، والموافقين لأول تقسيم أمثل لـ  $r_7$  للمعطى سابقاً.

تعاريف

### 1-1.15

يُبين أن المعادلة (4.15) تنتج من المعادلة (3.15) والحالة الابتدائية  $T(0) = 1$ .

### 2-1.15

يُبين، بإيراد أمثلة معاكسة، أن الاستراتيجية "الشرهة" التالية لا تحدد دوماً طريقة مثلى لتقطيع القضبان. عرّف



**كثافة density** قضيب طوله  $i$  على أنها  $p_i/i$ ، أي قيمته لكل بوصة. تُقطع الاستراتيجية الشرهة من قضيب طوله  $n$  قطعة أولى طولها  $i$ ، بحيث  $1 \leq i \leq n$ ، ذات كثافة عظمى. ثم تستمر بتطبيق الاستراتيجية الشرهة على القطعة المتبقية التي طولها  $n - i$ .

### 3-1.15

ادرس تعديلاً على مسألة تقطيع القضبان التي فيها، إضافة إلى سعر كل قضيب  $p_i$ ، كلفة ثابتة  $c$  لكل عملية قطع. إن الدخل المرتبط بكل حل هو الآن مجموع أسعار القطع مطروحاً منه تكاليف التقطيع. أعطِ خوارزمية برمجة ديناميكية تحل هذه المسألة المعدلة.

### 4-1.15

عدّل الإجراء MEMOIZED-CUT-ROD ليعيد ليس فقط القيمة وإنما الحل الفعلي أيضاً.

### 5-1.15

تُعرف أعداد فيوناتشي Fibonacci عودياً بالعلاقة (22.3). أعطِ خوارزمية برمجة ديناميكية تُنفذ بزمن  $O(n)$  لحساب عدد فيوناتشي من المرتبة  $n$ . ارسم بيان المسألة الجزئية. ما هو عدد العقد والوصلات في هذا البيان؟

## 2.15 جداء سلسلة من المصفوفات

مثالنا التالي على البرمجة الديناميكية هو خوارزمية حل مسألة جداء سلسلة من المصفوفات. ليكن لدينا متتالية (سلسلة) من  $n$  مصفوفة  $(A_1, A_2, \dots, A_n)$  وعلمنا إيجاد جدائهما، ونريد أن نحسب الجداء

$$A_1 A_2 \dots A_n. \quad (5.15)$$

يمكننا تقويم (حساب) العبارة (5.15) باستخدام الخوارزمية المعيارية لضرب أزواج المصفوفات باعتبارها مسألاً فرعياً، بعد أن نكون قد وضعناها ضمن أقواس لحل كل الجوانب المتعلقة بكيفية ضرب المصفوفات بعضها في بعض. ولما كان ضرب المصفوفات عملية تجميعية، فإن كل عمليات وضع الأقواس تؤدي إلى الجداء نفسه. نقول عن جداء مصفوفات أنه **كامل الأقواس fully parenthesized** إذا تكوّن من مصفوفة وحيدة أو من جداء مصفوفتين كاملتي الأقواس، محدودتين بأقواس. على سبيل المثال، إذا كانت سلسلة المصفوفات هي  $(A_1, A_2, A_3, A_4)$ ، فإن بإمكاننا أن نجعل الجداء  $A_1 A_2 A_3 A_4$  كامل الأقواس بخمس طرق متميزة هي:

$$(A_1(A_2(A_3A_4))) ,$$

$$(A_1((A_2A_3)A_4)) ,$$

$$((A_1A_2)(A_3A_4)) ,$$

$$((A_1(A_2A_3))A_4) ,$$

$$(((A_1A_2)A_3)A_4) .$$

وقد يكون لطريقة وضع الأقواس على سلسلة المصفوفات تأثيرٌ لافتٌ في تكلفة حساب جدائها. لندرس أولاً تكلفة جداء مصفوفتين. تُعطي الخوارزمية المعيارية من خلال شبه الرمان التالي، الذي يعمم الإجراء SQUARE-MATRIX-MULTIPLY من القطع 2.4، علماً أن الواصفات  $rows$  و  $columns$  هي عدد السطور وعدد الأعمدة في مصفوفة.

**MATRIX-MULTIPLY( $A, B$ )**

```

1  if  $A.columns \neq B.rows$ 
2      error "incompatible dimensions"
3  else let  $C$  be a new  $A.rows \times B.columns$  matrix
4      for  $i = 1$  to  $A.rows$ 
5          for  $j = 1$  to  $B.columns$ 
6               $c_{ij} = 0$ 
7              for  $k = 1$  to  $A.columns$ 
8                   $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
9  return  $C$ 
```

ولا نستطيع ضرب مصفوفتين  $A$  و  $B$  إلا إذا كانتا متوافقتين *compatible*: إذ لا بدّ من أن يكون عدد الأعمدة في  $A$  مساوياً لعدد السطور في  $B$ . فإذا كانت  $A$  مصفوفة  $p \times q$  وكانت  $B$  مصفوفة  $q \times r$ ، كانت المصفوفة الناتجة  $C$  هي مصفوفة  $p \times r$ . إن الزمن اللازم لحساب  $C$  محكوم بعدد الجداءات السلبية في السطر 8، وهو  $pqr$ . نبيّن فيما يلي التكاليف بدلالة عدد الجداءات السلبية.

ليبان التكاليف المختلفة التي يجليها مختلف أشكال وضع الأقواس لجداء المصفوفات، سندرس مسألة حساب جداء سلسلة ثلاث مصفوفات  $(A_1, A_2, A_3)$ ، ولنفترض أن أبعاد المصفوفات هي  $10 \times 100$  و  $100 \times 5$  و  $5 \times 50$ ، على التوالي. إذا أجرينا عملية الضرب بحسب الأقواس  $((A_1 A_2) A_3)$ ، فإننا نحتاج  $5000 = (10)(100)(5)$  عملية جداء سلّمي لحساب المصفوفة  $A_1 A_2$  التي هي مصفوفة  $10 \times 5$ ، ثم  $2500 = (10)(5)(50)$  عملية جداء سلّمي أخرى لحساب ناتج ضرب هذه المصفوفة بالمصفوفة  $A_3$ ، ويكون المجموع 7500 عملية جداء سلّمي. فلو أجرينا بدلاً من ذلك، عملية الضرب بناءً على وضع الأقواس  $(A_1 (A_2 A_3))$ ، لَلزِمنا  $25,000 = (100)(5)(50)$  عملية جداء سلّمي لحساب مصفوفة الجداء  $A_2 A_3$  مصفوفة  $100 \times 5$ ، إضافة إلى  $50,000 = (10)(100)(50)$  عملية جداء سلّمي لضرب الناتج في المصفوفة  $A_1$ ، ويكون عدد الجداءات السلبية الكلية هو 75,000. وبذلك يكون حساب الجداء بحسب وضعية الأقواس الأولى أسرع بعشر مرات.

ونصوغ مسألة جداء سلسلة من المصفوفات *matrix-chain multiplication problem* كما يلي:

إذا كان لدينا سلسلة  $(A_1, A_2, \dots, A_n)$  مؤلفة من  $n$  مصفوفة، حيث  $i = 1, 2, \dots, n$ ، فإن للمصفوفة  $A_i$  الأبعاد  $p_{i-1} \times p_i$ ، فالسلسلة هي وضع الأقواس الكاملة للجداء  $A_1 A_2 \dots A_n$  بحيث يكون عدد عمليات

الجداء السلمي أصغرًا.

لاحظ أننا في مسألة جداء سلسلة المصفوفات، لا تضرب المصفوفات فعليًا. وهدفنا فقط تحديد ترتيب لضرب المصفوفات بحيث تكون التكلفة صغرى. عموماً، فإن الوقت المصروف في تحديد الترتيب الأمثل يُعَوَّضُ بأكثر من ثمنه بالزمن المُدَّخَر لاحقاً، حين ننجز فعلياً ضرب المصفوفات (كما في تنفيذ 7500 عملية جداء سلمي فقط عوضاً عن 75,000 عملية).

### حساب عدد أوضاع الأقواس

قبل البدء بحل مسألة ضرب سلسلة المصفوفات بواسطة البرمجة الديناميكية، لنفنع أنفسنا بأن الاختيار الشامل لكل أوضاع الأقواس الممكنة لا يفضي إلى خوارزمية فعالة. لنعبر عن عدد بدائل وضع الأقواس لسلسلة من  $n$  مصفوفة بالرمز  $P(n)$ . فعندما تكون  $n = 1$  يكون لدينا مصفوفة واحدة، ومن ثم توجد طريقة واحدة لوضع كامل الأقواس لجداء المصفوفات. وعندما تكون  $n \geq 2$  فإن حاصل ضرب جداءتين جزئيتين كامليتي الأقواس للمصفوفات، ويمكن أن يحدث تفريق الجداءين الجزئيين بين المصفوفتين ذواتي الترتيب  $k$  و  $k + 1$  لأي قيمة  $k = 1, 2, \dots, n - 1$ . وبذلك نحصل على التكرار:

$$P(n) = \begin{cases} 1 & \text{if } n = 1. \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2. \end{cases} \quad (6.15)$$

وقد طُلب إليك في المسألة 4-12 البرهان على أن الحل لتكرار مشابه هو متتالية من **أعداد كاتلان** *Catalan numbers*، التي تتزايد كـ  $\Omega(4^n/n^{3/2})$ . ثمة تمثيل مشابه (انظر التمرين 3-2.15) بطلب البرهان على أن حل التكرار (6.15) هو  $\Omega(2^n)$ . وبذلك يكون عدد الحلول أسياً في  $n$ . ولذلك فإن طريقة البحث الشامل تؤدي إلى استراتيجيّة رديئة عند تحديد الوضع الأمثل للأقواس في جداء سلسلة مصفوفات.

### تطبيق البرمجة الديناميكية

سنستخدم طريقة البرمجة الديناميكية لتحديد الكيفية المثلى لوضع الأقواس لسلسلة مصفوفات. وللقيام بذلك، سننتج تنالي الخطوات الأربعة التي ذكرناها في بداية هذا الفصل:

1. وَصَفُ البنيان لـ **حلّ** أمثل.
2. عَرَفُ تَكَرُّراتٍ قيمة **حلّ** أمثل.
3. احسب قيمة **حلّ** أمثل.
4. أَنتِجْ **حلّاً** أمثل من للمعلومات المحسوبة

وستعرض لهذه الخطوات بالترتيب، مُبيّنين بوضوح كيف نطبق كل خطوة على المسألة.

### الخطوة 1: بنية الأقواس المثلى

لتنفيذ خطواتنا الأولى في نموذج البرمجة الديناميكية فإننا نوجد البنية الجزئية المثلى، ثم نستخدمها لبناء حل أمثل للمسألة من الحلول المثلى للمسائل الجزئية. وفي حالة مسألة جداء سلسلة المصفوفات، يمكننا إنجاز هذه الخطوة كما يلي. للتسهيل، سنعمد التدرين  $A_{1..r}$ ، حيث  $r \leq i$  للمصفوفة التي تنتج من حساب الجداء  $A_1 A_{i+1} \dots A_r$ . لاحظ أنه إذا كانت المسألة غير تافهة، أي  $r < i$ ، وجب أن يفرق أي وضع لأقواس الجداء  $A_1 A_{i+1} \dots A_r$  هذا الجداء بين المصفوفتين  $A_k$  و  $A_{k+1}$ ، حيث  $\parallel$  عدد صحيح ما في المجال  $i < k \leq i$ . أي إننا نحسب أولاً  $A_{1..r}$  و  $A_{k+1..r}$  لكل قيمة  $k$ ، ثم نضرب إحداها في الأخرى لحساب الجداء النهائي  $A_{1..r}$ . إن تكلفة الحساب بوضع الأقواس هذا هي مجموع تكلفة حساب المصفوفة  $A_{1..r}$  وحساب المصفوفة  $A_{k+1..r}$  إضافة إلى تكلفة ضرب إحداها بالأخرى.

البنية الجزئية المثلى لهذه المسألة هي كالتالي. افترض أن الوضع الأمثل للأقواس يفرق الجداء  $A_1 A_{i+1} \dots A_r$  بين المصفوفتين  $A_k$  و  $A_{k+1}$ . عندها يجب أن يكون وضع الأقواس للسلسلة الجزئية "البادئة"  $A_1 A_{i+1} \dots A_k$  ضمن الوضع الأمثل للأقواس  $A_1 A_{i+1} \dots A_r$  أمثل لـ  $A_1 A_{i+1} \dots A_k$ . لماذا؟ لأنه إذا كان هناك طريقة أقل تكلفة لوضع الأقواس للجداء  $A_1 A_{i+1} \dots A_k$ ، فإن تعويض هذه الأقواس في الوضع الأمثل للأقواس لحساب الجداء  $A_1 A_{i+1} \dots A_r$  سيُنتج وضعاً آخر للأقواس تكلفته أقل من التكلفة المثلى؛ وهذا تناقض. ونتحقق للملاحظة نفسها لوضع الأقواس للسلسلة الجزئية  $A_{k+1} A_{k+2} \dots A_r$  في الوضع الأمثل للأقواس في حساب الجداء  $A_1 A_{i+1} \dots A_r$ : إذ يجب أن يكون وضع الأقواس أمثل لوضع الأقواس  $A_{k+1} A_{k+2} \dots A_r$ .

الآن نستخدم بنية الجزئية المثلى لتبين أنه يمكننا بناء حل أمثل للمسألة من حلول مثلى للمسائل الجزئية. لقد رأينا أن أي حلٍّ مُنتسَخ غير تافه في مسألة جداء سلسلة المصفوفات يتطلب أن نفرق الجداء، وأن أي حلٍّ أمثل يتضمن في حد ذاته حلولاً مثلى لمُنتسَخات المسائل الجزئية. وبذلك، يمكننا بناء حلٍّ أمثل مُنتسَخ جداء سلسلة مصفوفات بتفريق المسألة إلى مسألتين جزئيتين (بوضع أقواس على نحو أمثل لكل من  $A_1 A_{i+1} \dots A_k$  و  $A_{k+1} A_{k+2} \dots A_r$ )، ثم إيجاد حلول مثلى لمُنتسَخات المسائل الجزئية، ثم بضم هذه الحلول المثلى للمسائل الجزئية. يجب أن نتأكد، حين نبحث عن المكان الصحيح لتفريق الجداء، أننا أخذنا كل أماكن التفريق المحتملة بالحسبان، بحيث نتأكد أننا نحزّنا الوضع الأمثل.

### الخطوة 2: حلٌّ عَوْدِي

بعد ذلك، نحدّد تكلفة الحل الأمثل عودياً بدلالة الحلول المثلى للمسائل الجزئية. وفي حالة مسألة جداء سلسلة

المصفوفات، تعتبر مسائلنا الجزئية هي مسائل تحديد التكلفة الصغرى لوضع الأقواس  $A_1 A_{i+1} \dots A_j$  حيث  $1 \leq i \leq j \leq n$ . ليكن  $m[i, j]$  أصغر عدد لعمليات الجداء السلمية اللازمة لحساب المصفوفة  $A_{i..j}$ ؛ عندها، لحل المسألة الكلية، ستكون تكلفة طريقة التكلفة الدنيا لحساب  $A_{1..n}$  هي  $m[1, n]$ . ويمكننا تحديد  $m[i, j]$  عودياً كما يلي. إذا كانت  $i = j$  فالمسألة تافهة؛ وتتكون السلسلة من مصفوفة واحدة  $A_{i..i} = A_i$ ، وبذلك لا نحتاج إلى أي عمليات جداء سلمية لحساب الجداء. وبذلك يكون  $m[i, i] = 0$  حيث  $i = 1, 2, \dots, n$ . ولحساب  $m[i, j]$  عندما تكون  $j < i$ ، نستفيد من بنية الحل الأمثل في الخطوة 1. لنفترض أن وضع الأقواس الأمثل يفرق الجداء  $A_1 A_{i+1} \dots A_j$  بين  $A_k$  و  $A_{k+1}$  حيث  $k < j \leq i$ . حينها يكون  $m[i, j]$  مساوياً للتكلفة الصغرى لحساب الجداءات الجزئية  $A_{i..k}$  و  $A_{k+1..j}$ ، إضافة إلى تكلفة ضرب إحدى هاتين المصفوفتين بالأخرى. لتتذكر أن أبعاد كل مصفوفة  $A_i$  هو  $p_{i-1} \times p_i$ ، فترى أن حساب مصفوفة الجداء  $A_{i..k} A_{k+1..j}$  يتطلب إجراء  $p_{i-1} p_k p_j$  عملية جداء سلمية. وبذلك يصبح لدينا:

$$m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j .$$

نفترض هذه المعادلة العودية أننا نعلم قيمة  $k$ ، ولكن الأمر ليس كذلك. على أن لمة  $i - j$  قيمة ممكنة فقط لـ  $k$ ، هي  $1 - j \leq k \leq i - 1$ . ولأن وضع الأقواس الأمثل يجب أن يستخدم قيمة واحدة فقط من قيم  $k$ ، فما علينا إلا أن ندرّج فيها جميعها لإيجاد الأفضل. وبذلك يصبح تعريفنا العودي للتكلفة الصغرى لوضع أقواس الجداء  $A_1 A_{i+1} \dots A_j$  كما يلي:

$$m[i, j] = \begin{cases} 0 & \text{if } i = j , \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\} & \text{if } i < j . \end{cases} \quad (7.15)$$

تعطي القيم  $m[i, j]$  تكاليف الحلول المثلى للمسائل الجزئية، ولكنها لا تعطي كل المعلومات التي نحتاج إليها لبناء حل أمثل. ونستعين على ذلك بأن نعرّف  $s[i, j]$  على أنها قيمة  $k$  التي نفرق عندها الجداء  $A_1 A_{i+1} \dots A_j$  للحصول على وضع الأقواس الأمثل. أي إن  $s[i, j]$  تساوي قيمة  $k$  بحيث  $m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$ .

### الخطوة 3: حساب التكاليف المثلى

عند هذه النقطة، من السهل كتابة خوارزمية عودية تعتمد على العلاقة العودية (7.15) لحساب التكلفة الصغرى  $m[1, n]$  للجداء  $A_1 A_2 \dots A_n$ . وحسيماً رأينا في مسألة تقطيع القضبان، وكما سنرى في المقطع 3.15، فإن هذه الخوارزمية تستغرق زمناً أسياً، وهو ليس أفضل من طريقة البحث الشامل لفحص كل طرق وضع الأقواس لحساب الجداء.

لاحظ أن لدينا عدداً قليلاً نسبياً من مسائل جزئية متمايزة: مسألة واحدة لكل خيار لـ  $i$  و  $j$  بحيث

يكون  $1 \leq i \leq j \leq n$ ، أو تعقيدًا كافيًا  $\Theta(n^2)$ ، يمكن أن تصادف خوارزمية عودية كل مسألة

جزئية عدّة مرات في الفروع المختلفة لشجرها العودية. إن خاصية تداخل المسائل الجزئية هذه هي سمة مميزة ثانية لقابلية تطبيق البرمجة الديناميكية (السمة المميزة الأولى هي البنية الجزئية المثلى).

عوضًا عن حساب الحل للعلاقة العودية (7.15) عوديًا، فإننا نحسب التكلفة المثلى باستخدام نهج صعودي يعتمد جدولًا. (سنعرض النهج التّرجولي الموافق باستخدام الاستدّكار في المقطع 3.15).

وسننجز الطريقة الصعودية الجدولية في الإجراء MATRIX-CHAIN-ORDER الذي يظهر لاحقًا. يفترض هذا الإجراء أن يُمدد المصفوفة  $A_i$  هو  $p_{i-1} \times p_i$  حيث  $i = 1, 2, \dots, n$ . ودخل الإجراء هو المتتالية  $m[1..n, 1..n]$ ، حيث  $p = (p_0, p_1, \dots, p_n)$ . ويستخدم الإجراء جدولًا مساعدًا  $s[1..n-1, 2..n]$  لتسجيل الدليل  $k$  الذي كان قد أُخِز لحزن التكاليف  $m[i, j]$ ، وجدولًا مساعدًا آخر  $s[1..n-1, 2..n]$  لتسجيل الدليل  $k$  الذي كان قد أُخِز التكلفة المثلى في حساب  $m[i, j]$ . ونستخدم الجدول  $s$  لبناء الحل الأمثل.

لتنجيز النهج الصعودي، علينا أن نحدد عناصر الجدول المُستخدمة في حساب  $m[i, j]$ . تبين للمعادلة (7.15) أن التكلفة  $m[i, j]$  لحساب جداء سلسلة المصفوفات المكون من  $j - i + 1$  مصفوفة يعتمد فقط على تكاليف حساب جداء سلسلة مصفوفات عددها أقل من  $j - i + 1$ . أي لكل  $k = i, i + 1, \dots, j - 1$ ، فإن المصفوفة  $A_k$  هي جداء  $j - i + 1 < k - i + 1$  مصفوفة، والمصفوفة  $A_{k+1}$  هي جداء  $j - i + 1 < j - k$  مصفوفة. وبذلك، يجب أن تملأ الخوارزمية الجدول بقيم  $m$  على نحو يتوافق مع حل مسألة وضع الأقواس على سلسلة مصفوفات ذات طول متزايد. وفيما يتعلق بالمسألة الجزئية لوضع الأقواس على نحو أمثل لسلسلة المصفوفات  $A_1 A_{i+1} \dots A_j$ ، نفترض أن حجم المسألة الجزئية هو طول السلسلة  $j - i + 1$ .

#### MATRIX-CHAIN-ORDER(p)

```

1   $n = p.length - 1$ 
2  let  $m[1..n, 1..n]$  and  $s[1..n-1, 2..n]$  be new tables
3  for  $i = 1$  to  $n$ 
4       $m[i, i] = 0$ 
5  for  $l = 2$  to  $n$  //  $l$  is the chain length.
6      for  $i = 1$  to  $n - l + 1$ 
7           $j = i + l - 1$ 
8           $m[i, j] = \infty$ 
9          for  $k = i$  to  $j - 1$ 
10              $q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$ 
11             if  $q < m[i, j]$ 
12                  $m[i, j] = q$ 
13                  $s[i, j] = k$ 
14  return  $m$  and  $s$ 
```

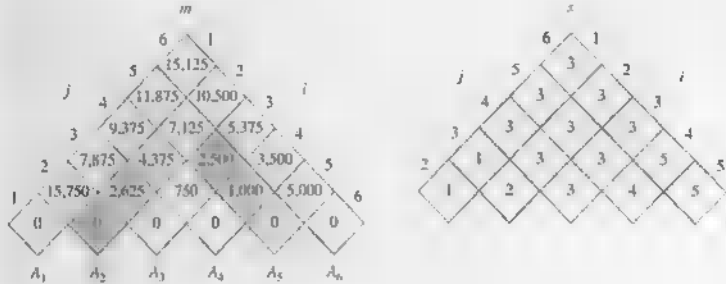
نحسب الخوارزمية أولاً  $m[i, i] = 0$  للقيم  $i = 1, 2, \dots, n$  (التكاليف الصغرى لسلاسل بطول 1) في السطرين 3-4. بعد ذلك نستخدم العلاقة التكرارية (7.15) لحساب  $m[i, i+1]$  لقيم  $i = 1, 2, \dots, n-1$  (التكاليف الصغرى لسلاسل بطول 2) خلال التنفيذ الأول لحلقة for في السطور 5-13. وخلال التنفيذ الثاني للحلقة نحسب  $m[i, i+2]$  لقيم  $i = 1, 2, \dots, n-2$  (التكاليف الصغرى لسلاسل بطول 3)، وهكذا. في كل خطوة، تعتمد التكاليف  $m[i, j]$  المحسوبة في السطور 10-13 فقط على عناصر الجدول  $m[i, k]$  و  $m[k+1, j]$  المحسوبة سلفاً.

يبين الشكل 5.15 هذه الإجرائية على سلسلة من  $n = 6$  مصفوفات. ولأننا عرّفنا  $m[i, j]$  فقط في حالة  $j \leq i$ ، فإننا نستخدم فقط جزء الجدول  $m$  الذي يقع فوق القطر تماماً. يبين الشكل الجدول مدوّراً لجعل القطر الرئيسي يبدو أفقياً. جرى سرد سلسلة المصفوفات في الأسفل. وباستخدام هذا المخطط، يمكن إيجاد التكلفة الصغرى  $m[i, j]$  لجداء السلسلة الجزئية من المصفوفات  $A_1 A_{i+1} \dots A_j$  عند تقاطع الخط الذي يبدأ من  $A_i$  باتجاه الشمال الشرقي مع الخط الذي يبدأ من  $A_j$  باتجاه الشمال الغربي. يتضمن كل سطر أفقي في الجدول القيم من أجل سلاسل المصفوفات ذات الطول نفسه. بحسب الإجراء MATRIX-CHAIN-ORDER السطور من الأسفل إلى الأعلى ومن اليسار إلى اليمين ضمن كل سطر. وهو يحسب كل العناصر  $m[i, j]$  باستخدام الجداء  $p_{i-1} p_k p_j$  لقيم  $k = i, i+1, \dots, j-1$  ولكل العناصر الجنوبية الغربية والجنوبية الشرقية من  $m[i, j]$ .

تبيّن معانيّة بسيطة لبنية الحلقات المتداخلة في MATRIX-CHAIN-ORDER أن زمن تنفيذ الخوارزمية هو  $O(n^3)$ . إذ هناك ثلاث حلقات متداخلة، وكل دليل حنقة ( $i$  و  $j$  و  $k$ ) يأخذ  $n-1$  قيمة على الأكثر. يُطلب إليك في التمرين 5-2.15 أن تبين أن زمن تنفيذ هذه الخوارزمية هو في الحقيقة  $\Omega(n^3)$  أيضاً. تتطلب الخوارزمية  $\Theta(n^2)$  مكاناً لحزن الجدولين  $m$  و  $s$ . وبذلك، فإن الخوارزمية MATRIX-CHAIN-ORDER أكثر فعالية بكثير من الطريقة التي زعمنا أسي، والتي تحصى كل أوضاع الأقواس الممكنة وتفحص كلاً منها.

#### الخطوة 4: بناء حلّ الأمثل

مع أن الخوارزمية MATRIX-CHAIN-ORDER تحدد العدد الأمثل للجداءات السلبية اللازمة لحساب جداء سلسلة مصفوفات، فهي لا تبيّن مباشرة كيف تجري عملية ضرب المصفوفات. لكن الجدول  $s[1..n-1, 2..n]$  يزودنا بالمعلومات اللازمة للقيام بذلك. يسجل كل عنصر  $s[i, j]$  قيمة  $k$  بحيث يفرض الوضخ الأمثل لأقواس الجداء  $A_i A_{i+1} \dots A_j$  هذا الجداء بين  $A_k$  و  $A_{k+1}$ . وبذلك نعلم أن جداء مصفوفات النهاية في حساب  $A_{1..n}$  أمثلًا هو  $A_{1..s[1,n]} A_{s[1,n]+1..n}$ . ويمكن حساب جداءات المصفوفات السابقة عودتًا، لأن  $s[1, s[1, n]]$  تحدد آخر جداء مصفوفات عند حساب  $A_{1..s[1,n]}$  وتحدد  $s[s[1, n]+1, n]$  وآخر جداء مصفوفات عند حساب  $A_{s[1,n]+1..n}$ . يطبع الإجراء العودي التالي وضخًا أمثلًا للأقواس



**الشكل 5.15** الجداول  $m$  و  $s$  محسوبة بالخوارزمية MATRIX-CHAIN-ORDER في حالة  $n = 6$  وأبعاد المصفوفات التالية:

$A_6$	$A_5$	$A_4$	$A_3$	$A_2$	$A_1$	المصفوفة
$20 \times 25$	$10 \times 20$	$5 \times 10$	$15 \times 5$	$35 \times 15$	$30 \times 35$	أبعادها

جرى تدوير الجداول بحيث يظهر القطر الرئيسي أفقيًا. لا نستخدم إلا القطر الرئيسي ولتلك الذي فوقه في الجدول  $m$ ، ولتلك العلوي فقط في الجدول  $s$ . العدد الأصغر للحداءات السلمية لضرب ست مصفوفات هو  $m[1,6] = 15,125$ . ومن العناصر الفارقة، نؤخذ الأزواج التي لها التظليل نفسه منا في السطر 10 عند حساب

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + (35)(15)(20) = 13000, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + (35)(5)(20) = 7125, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + (35)(10)(20) = 11375 \end{cases}$$

$$= 7125.$$

في  $(A_i, A_{i+1}, \dots, A_j)$ ، في حال أعطينا الجدول  $s$  المحسوب بواسطة MATRIX-CHAIN-ORDER والدليلين  $i$  و  $j$ . ويطلع الاستدعاء الأولي للإجرائية  $\text{PRINT-OPTIMAL-PARENS}(s, 1, n)$  وضعا أمثل لأقواس الجداء  $\langle A_1, A_2, \dots, A_n \rangle$ .

$\text{PRINT-OPTIMAL-PARENS}(s, i, j)$

```

1 if  $i == j$ 
2   print " $A_i$ "
3 else print "("
4   PRINT-OPTIMAL-PARENS( $s, i, s[i, j]$ )
5   PRINT-OPTIMAL-PARENS( $s, s[i, j] + 1, j$ )
6   print ")"
```

في المثال المبين بالشكل 5.15 يطلع الاستدعاء  $\text{PRINT-OPTIMAL-PARENS}(s, 1, 6)$  الأقواس التالية:

$$((A_1(A_2A_3))((A_4A_5)A_6))$$



## تعاريف

## 1-2.15

أوجد وضع الأقواس الأمثل لجداء سلسلة مصفوفات، تنالي أبعادها هو: (5, 10, 3, 12, 5, 50, 6).

## 2-2.15

أعط الخوارزمية العودية  $MATRIX-CHAIN-MULTIPLY(A, s, i, j)$  التي تنجز فعليًا الجداء الأمثل لسلسلة مصفوفات، إذا كان لدينا تنالي للمصفوفات  $(A_1, A_2, \dots, A_n)$ ، والجندول  $s$  المحسوب بالخوارزمية  $MATRIX-CHAIN-ORDER$  والدليلان  $i$  و  $j$ . (يمكن أن يكون الاستدعاء الأول لهذه الخوارزمية هو:  $MATRIX-CHAIN-MULTIPLY(A, s, 1, n)$ ).

## 3-2.15

استخدم طريقة التعويض لتبين أن حل العلاقة التكرارية (6.15) هو  $\Omega(2^n)$ .

## 4-2.15

صِف بيان المسألة الجزئية لضرب سلسلة المصفوفات، بسلسلة دخل طولها  $n$ . كم عدد العقد فيه؟ وكم عدد الوصلات فيه؟ وما هي هذه الوصلات؟

## 5-2.15

ليكن  $R(i, j)$  عدد المرات التي تعود فيها إلى عنصر الجدول  $m[i, j]$  عند حساب عناصر الجدول الأخرى في الاستدعاء  $MATRIX-CHAIN-ORDER$ . بين أن العدد الكلي للمرات التي تعود فيها إلى كامل الجدول هو:

$$\sum_{i=1}^n \sum_{j=i}^n R(i, j) = \frac{n^3 - n}{3}.$$

(لملح: يمكن الاستفادة من المساواة (أ.3).)

## 6-2.15

بين أن وضع كامل الأقواس لتعبير من  $n$  عنصراً يتطلب تماماً  $n - 1$  زوجاً من الأقواس.

## 3.15 عناصر البرمجة الديناميكية

مع أننا عملنا حتى الآن على مثالين على طريقة البرمجة الديناميكية، فربما ما زلت تتساءل أين يمكن تطبيق هذه الطريقة. فمن وجهة نظر هندسية تتساءل: متى يتعين علينا البحث عن حلٍّ مسألة بالبرمجة الديناميكية؟ في هذا المقطع، سندرس المكوّنَيْن الرئيسيين اللذين يجب أن يتوفرا في مسألة الأمثلة لكي يكون بالإمكان تطبيق البرمجة الديناميكية: بنية جزئية مُثلى، ومسائل جزئية متراكبة. سنعود أيضاً وناقش بتفصيل أكبر كيف يمكن أن يساعدنا الاستدكار memoization للإفادة من خاصية تراكب المسائل الجزئية في نصح عودي نزولي.

### البنية الجزئية المثلى

تكمّن الخطوة الأولى -في حل مسألة أمثلة بالبرمجة الديناميكية- في توصيف بنية حل أمثل. نذكر أن مسألة ما، تُظهر بنية جزئية مثلى *optimal substructure* إذا تضمن الحل الأمثل للمسألة حلولاً مثلى لمسائل جزئية. فكلما أبدت مسألة ما بنية جزئية مثلى، فهذا دليل جيد على إمكان تطبيق البرمجة الديناميكية. (ويع ذلك، فإن هذا يمكن أن يعني أيضاً إمكان تطبيق استراتيجيات شجرة. انظر الفصل 16). في البرمجة الديناميكية، نبني الحل الأمثل للمسألة من الحلول المثلى للمسائل الجزئية. ومن ثمّ فعلينا التأكيد أن مجال المسائل الجزئية التي ندرسها يتضمن تلك المستخدمة في الحل الأمثل.

اكتشفنا حتى الآن البنية الجزئية المثلى في كلتا المسائلتين اللتين درسناهما في هذا الفصل. ففي المقطع 1.15، لاحظنا أن الطريقة المثلى لقطع قضيب طوله  $n$  (إذا كان هناك قطع أصلاً) تتطلب تقطيعاً أمثل لقطعتين نتجتا عن أول قطع. ولاحظنا في المقطع 2.15 أن وضع الأقواس الأمثل لـ  $A_1 A_{i+1} \dots A_j$  الذي يفرق الجداء بين  $A_k$  و  $A_{k+1}$  يتضمن حلولاً مثلى لمسائل وضع الأقواس لكل من  $A_1 A_{i+1} \dots A_k$  و  $A_{k+1} A_{k+2} \dots A_j$ .

وستجد نفسك تتعقب نموذجاً مشتركاً في اكتشاف بنية جزئية مثلى:

1. سيتبين لك أن حل المسألة يتضمن إجراء عملية اختيار، مثل اختيار القطع الأولي في قضيب أو اختيار الدليل الذي نفرق عنده سلسلة المصفوفات. إن إجراء هذا الاختيار يقي مسألة جزئية أو أكثر يجب حلها.
2. سنفترض أنه في مسألة معينة، سيكون لديك الخيار الذي يقود إلى حل أمثل. لن يكون عليك أن نحتم بعدد بكيفية تحديد هذا الخيار، إنك نفترض أنه قد أعطي إليك.
3. إذا أعطيت هذا الخيار، عليك أن تحدد أي للمسائل الجزئية ستنشأ، وكيف ستوصف أمثلاً فضاء المسائل الجزئية الناتجة.
4. سيتبين لك أن حلول المسائل الجزئية المستخدمة ضمن الحل الأمثل للمسألة يجب أن تكون هي نفسها مثلى، باستخدام تقنية "قصّ والصقّ". وبإمكانك إجراء ذلك بافتراض أن كلاً من حلول للمسائل الجزئية غير أمثلي، ثم الوصول إلى تناقض. وبوجه خاص، ستري أنك "بقصّ" حلول للمسائل الجزئية غير المثلى، و"الصقّ" الحل الأمثل، ستصل إلى حل أفضل للمسألة الأصلية، وبذلك تُناقض افتراضك بأنه كان لديك سابقاً حل أمثل. وإذا أعطيت حل أمثل أكثر من مسألة جزئية واحدة، كانت هذه للمسائل الجزئية متشابهة جداً إلى درجة أنه يمكن، بتقليل من الجهد، تعديل عملية القصّ واللصق لإحداها وتطبيقها على المسائل الأخرى.

ولتوصيف فضاء المسائل الجزئية، فإن القاعدة الذهبية تقول بأن نحاول جعل هذا الفضاء سهلاً قدر الإمكان، ثم توسيعه بقدر الحاجة إلى ذلك. على سبيل المثال، كان فضاء المسائل الجزئية الذي أعدهنا بالحساب في مسألة تقطيع القضبان يتضمن مسائل تقطيع أمثل لقضيب طوله  $i$ ، لكل قيم الطول  $i$ . وكان هذا الفضاء جيد الأداء، ولم يكن هناك داعٍ لتجريب فضاء مسائل جزئية أكثر عمومية.

وبالعكس، افترض أننا حاولنا أن نُقصر فضاء المسائل الجزئية في مسألة جداء سلسلة مصفوفات على جداء مصفوفات من الشكل  $A_1 A_2 \dots A_k$ . كما في السابق، يجب أن نفرق وضع الأقواس الأمثل لهذا الجداء بين المصفوفتين  $A_k$  و  $A_{k+1}$ ، لقيمة ما  $k$  بحيث  $1 \leq k \leq z$ . وما لم نضمن أن  $k$  تساوي دائماً  $z-1$ ، سنجد أن لدينا مسائل جزئية من الشكل  $A_1 A_2 \dots A_k$  و  $A_{k+1} A_{k+2} \dots A_z$ ، وأن المسألة الجزئية الأخيرة ليست من الشكل  $A_1 A_2 \dots A_z$ . وفي هذه المسألة، كان يلزمنا أن نسمح لمسائلنا الجزئية أن تتغير "من الطرفين"، أي أن نسمح ل  $i$  و  $z$  بالتغير في المسألة الجزئية  $A_1 A_{i+1} \dots A_z$ .

يمكن أن تتغير البنية الجزئية المثلى عبر مجالات المسألة بطريقتين:

1. ما هو عدد المسائل الجزئية التي يستخدمها الحل الأمثل للمسألة الأصلية، و
2. ما هو عدد الخيارات لدينا لتحديد المسألة أو لمسائل الجزئية التي علينا استخدامها في الحل الأمثل.

في مسألة تقطيع القضبان، نستخدم الحل الأمثل لتقطيع قضيب طوله  $n$  مسألة جزئية وحيدة (حجمها  $n-1$ )، ولكن علينا أن نأخذ بالحساب  $n$  خياراً ل  $i$  لتحديد أيها يعطي حلاً أمثل. تمثل مسألة جداء سلسلة المصفوفات للسلسلة الجزئية  $A_1 A_{i+1} \dots A_z$  مثلاً له مسألتان جزئيتان و  $i-z$  خياراً. في حالة مصفوفة معلومة  $A_k$  التي نفرق عندها الجداء، سيكون لدينا مسألتان جزئيتان: وضع الأقواس للجداء  $A_1 A_{i+1} \dots A_k$  ووضعها للجداء  $A_{k+1} A_{k+2} \dots A_z$ ، وعلينا أن نحل كل منهما حلاً أمثل. وحلنا نجد الحلول المثلى للمسائل الجزئية، نختار الدليل  $k$  من بين الـ  $i-z$  دليلاً مُرشحاً.

وعلى نحو غير رسمي، يعتمد زمن تنفيذ خوارزمية البرمجة الديناميكية على جداء عاملين: عدد المسائل الجزئية الكلية، وعدد الخيارات التي ننظر فيها لكل مسألة جزئية. ففي مسألة تقطيع القضبان، كان عدد المسائل الجزئية الكلية  $\Theta(n)$ ، وعلينا أن نفحص  $n$  خياراً على الأكثر لكل منها، وهذا يعطي زمن تنفيذ  $O(n^2)$ . وفي حالة جداء سلسلة المصفوفات، كان عدد المسائل الجزئية الكلية  $\Theta(n^2)$ ، ولكل منها  $n-1$  خياراً على الأكثر، وبذلك يكون زمن التنفيذ  $O(n^3)$  (فعلياً زمن التنفيذ  $\Theta(n^3)$ ، وفقاً للتمرين 2.15-5).

يُعطي بيان المسائل الجزئية، عادة، طريقة بديلة لإيجاز التحليل نفسه؛ إذ توافق كل عقدة مسألة جزئية، وخيارات أي مسألة جزئية هي الوصلات المرتبطة بتلك المسألة الجزئية. تذكر أنه في مسألة تقطيع القضبان، يتضمن بيان المسألة الجزئية  $n$  عقدة، و  $n$  وصلة على الأكثر لكل عقدة، وهذا يعطي زمن تنفيذ  $O(n^2)$ . وفي حالة مسألة جداء المصفوفات، لو كان علينا رسم بيان المسائل الجزئية، لتضمّن  $\Theta(n^2)$  عقدة، ولكان

لكل عقدة درجة تساوي على الأكثر  $n - 1$ ، وهذا يعطي ما مجموعه  $O(n^3)$  عقدة ووصلة. غالبًا ما تُستخدم البرمجة الديناميكية البنية الجزئية المثلى بطريقة صعودية. أي إننا نجد أولاً حلولاً مثلى لمسائل جزئية، وبعد حل المسائل الجزئية نجد حلاً مثلياً للمسألة. ويستلزم إيجاد حل أمثل للمسألة، اختيار المسألة الجزئية التي سنستخدمها في حل المسألة من بين المسائل الجزئية. إن تكلفة حل المسألة تساوي عادة تكاليف المسائل الجزئية إضافة إلى تكلفة تُعزى مباشرة إلى الاختيار نفسه. على سبيل المثال، في مسألة تقطيع القضبان، قمنا أولاً بحل المسائل الجزئية لإيجاد الطرق المثلى لتقطيع القضبان ذات الطول  $i$  لقيم  $i = 0, 1, \dots, n - 1$  ثم حددنا أي المسائل الجزئية أعطت حلاً أمثل لقضيب طوله  $n$  باستخدام المعادلة (2.15). إن تكلفة الخيار نفسه هو الحد  $p_i$  في المعادلة (2.15). وفي مسألة جداء سلسلة المصفوفات، حددنا وضع الأقواس الأمثل للسلسلة الجزئية  $A_i A_{i+1} \dots A_j$ ، ثم اخترنا المصفوفة  $A_k$  التي نقرق الجداء عندها. أما التكلفة التي نعزوها إلى الاختيار نفسه فهي الحد  $p_{i-1} p_k p_j$ .

سندرس في الفصل 16 "الخوارزميات الشجرة"، التي تشابه كثيرًا البرمجة الديناميكية. وعلى وجه الخصوص، فإن للمسائل التي تنطبق عليها الخوارزميات الشجرة بنية جزئية مثلى. ومن الفروق الرئيسية بين الخوارزميات الشجرة والبرمجة الديناميكية أنه عوضًا عن إيجاد الحلول المثلى للمسائل الجزئية أولاً، ثم الاختيار عن علم، تقوم الخوارزميات الشجرة أولاً باختيار "شجرة" - الخيار الذي يبدو أنه الأفضل في ذلك الوقت - ثم تحل المسألة الجزئية الناتجة، دون الالتفات إلى حل كل المسائل الجزئية الأصغر المحتملة ذات الصلة. وما يثير الدهشة فعلاً أن هذه الاستراتيجية تعمل بنجاح في بعض الأحيان!

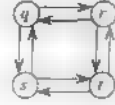
### نقاط دقيقة

ينبغي ألا نفترض أن البنية الجزئية المثلى معقّدة، في حين لا يكون الأمر كذلك. لندرس المسائلين التاليين، اللتين لدينا فيهما بيان موجه  $G = (V, E)$  والعقد  $u, v \in V$ .

**أقصر مسار غير مثقل<sup>3</sup>:** أوجد مسارًا من  $u$  إلى  $v$  مؤلفًا من أقل عدد من الوصلات. يجب أن يكون هذا المسار بسيطًا، لأن حذف حلقة cycle من المسار يؤلّد طريقًا بوصلات أقل.

**أطول مسار بسيط غير مثقل:** أوجد مسارًا بسيطًا من  $u$  إلى  $v$  يتكون من معظم الوصلات. نحتاج إلى إدخال مطلب البساطة لأننا بغیر ذلك يمكن أن نجتاز حلقةً بقدر ما نريد من المرات لنشغّل مساراتٍ بعدد وصلات لا على التعيين.

<sup>3</sup> نستخدم التعبير "غير مثقل unweighted" لتمييز هذه المسألة عن تلك التي توجد أقصر مسار مع وصلات مثقلة weighted، التي ستناولها في الفصلين 24 و 25. يمكننا توظيف تقنية البحث عرضًا أولاً breadth-first search technique المستخدمة في الفصل 22 لحل المسألة غير المثقلة.



**الشكل 6.15** بيان موجه بين أنه لا توجد بنية جزئية مثلى مسألة إيجاد أطول مسار بسيط في بيان موجه غير مثقل. المسار  $q \rightarrow r \rightarrow t$  هو أطول مسار بسيط من  $q$  إلى  $t$ ، إلا أن المسار الجزئي  $q \rightarrow r$  ليس أطول مسار بسيط من  $q$  إلى  $r$ ، كما أن المسار الجزئي  $r \rightarrow t$  ليس أطول مسار بسيط من  $r$  إلى  $t$ .

تُبدى مسألة أقصر طريق غير مثقل بنية مثلى كما يلي: افترض أن  $u \neq v$ ، بحيث لا تكون المسألة تافهة. بعدها، يجب أن يتضمن كل مسار  $p$  من  $u$  إلى  $v$  عقدة وسيطة ولكن  $uv$  (لاحظ أن  $uv$  يمكن أن تكون  $u$  أو  $v$ ). وبذلك يمكننا تقسيم المسار  $u \xrightarrow{p} v$  إلى المسارات الجزئية  $u \xrightarrow{p_1} w$  و  $w \xrightarrow{p_2} v$ . ومن الواضح أن عدد الوصلات  $p$  يساوي مجموع عدد الوصلات في  $p_1$  و  $p_2$ . ندعي أنه إذا كانت  $p$  مسارًا أمثل (أقصر مسار) من  $u$  إلى  $v$  فإن  $p_1$  يجب أن تكون أقصر مسار من  $u$  إلى  $w$ . لماذا؟ نستخدم طريقة المحاكمة "قص والصق": فلو كان هناك مسار آخر، وليكن  $p'_1$ ، من  $u$  إلى  $w$  بوصلات أقل من  $p_1$ ، لأمكننا إذا قص  $p_1$  ولصق  $p'_1$  لتوليد مسار  $u \xrightarrow{p'_1} w \xrightarrow{p_2} v$  بوصلات أقل من  $p$ . وهذا يناقض كون  $p$  مثاليًا. وبالمثل، يجب أن تكون  $p_2$  أقصر مسار من  $w$  إلى  $v$ . وبذلك، يمكن إيجاد أقصر مسار من  $u$  إلى  $v$  بأخذ كل العقد الوسيطة  $w$  بالاعتبار، وإيجاد أقصر مسار من  $u$  إلى  $w$  وأقصر مسار من  $w$  إلى  $v$ ، واختيار عقدة وسيطة  $w$  تعطي أقصر مسار كلي. نستخدم في المقطع 2.25 شكلاً مختلفاً لهذه الملاحظة المتعلقة بالبنية الجزئية المثلى لإيجاد أقصر مسار بين كل زوجين من العقد على بيان موجه مثقل.

قد يكون من المفري افتراض أن مسألة إيجاد أطول مسار بسيط غير مثقل تبدي بنية جزئية مثلى أيضاً. وبعد هذا، إذا حللنا أطول مسار بسيط  $u \xrightarrow{p} v$  إلى مسارتين جزئيتين  $u \xrightarrow{p_1} w$  و  $w \xrightarrow{p_2} v$ ، عندها ليس من الواجب أن تكون  $p_1$  أطول مسار بسيط بين  $u$  و  $w$ ، وأن تكون  $p_2$  أطول مسار بسيط بين  $w$  و  $v$ ؟ الجواب لا! يبين الشكل 6.15 مثالاً على ذلك. لتأخذ للمسار  $q \rightarrow r \rightarrow t$  وهو أطول مسار بسيط من  $q$  إلى  $t$ . هل  $q \rightarrow r$  هي أطول مسار بسيط من  $q$  إلى  $r$ ؟ الجواب لا، لأن المسار  $q \rightarrow s \rightarrow r$  هو أطول مسار بسيط من  $q$  إلى  $r$ . هل  $r \rightarrow t$  هي أطول مسار بسيط من  $r$  إلى  $t$ ؟ الجواب لا أيضاً، لأن المسار  $r \rightarrow q \rightarrow s \rightarrow t$  هو أطول مسار بسيط من  $r$  إلى  $t$ .

يبين هذا لكأن أنه في حالة أطول مسارات بسيطة، لا تفتقر المسألة إلى بنية جزئية مثلى فقط؛ وإنما لا يمكننا بالضرورة تجميع حل "شرعي" للمسألة من حلول لمسائل جزئية. إذا صمّمنا المسارتين البسيطتين الطويلتين:  $q \rightarrow r \rightarrow t$  و  $q \rightarrow s \rightarrow t$  حصلنا على  $q \rightarrow r \rightarrow t$  و  $q \rightarrow s \rightarrow t$  و  $q \rightarrow r \rightarrow s \rightarrow t$  و  $q \rightarrow s \rightarrow r \rightarrow t$  و  $q \rightarrow r \rightarrow s \rightarrow t$  و  $q \rightarrow s \rightarrow r \rightarrow t$ .

وهو مسار ليس بسيطًا. في الواقع، لا يبدو أن مسألة إيجاد أطول مسار بسيط غير مثقل تمتلك أي نوع من البنى الجزئية المثلى. ولم يُفكر قطُّ على أي خوارزمية تعتمد على برمجة ديناميكية فعالة لحل هذه المسألة. والحقيقة أن هذه المسألة تامة غير حدودية NP-complete، وهذا يعني - كما سنرى في الفصل 34 - أنه ليس من المحتمل أن نجد طريقًا لحلها بزمَن كثير حدودي.

لماذا كانت البنية الجزئية لأطول مسار بسيط مختلفة جدًا عن البنية الجزئية لأقصر مسار؟ ومع أن حلَّ مسألة أطول المسارات وأقصرها يتطلب استخدام مسألتين جزئيتين، فإن المسائل الجزئية المستخدمة في إيجاد أطول مسار بسيط ليست مسائل مستقلة independent بعضها عن بعض، في حين أنها مستقلة في حالة أقصر المسارات. ماذا نقصد بكون المسائل الجزئية مستقلة؟ نقصد بذلك أن حل إحدى المسائل الجزئية لا يؤثر في حل مسألة جزئية أخرى للمسألة الأصلية نفسها. على سبيل المثال، في الشكل 6.15 لدينا مسألة إيجاد أطول مسار بسيط من  $q$  إلى  $t$  ولها مسألتان جزئيتان: إيجاد أطول مسار بسيط من  $q$  إلى  $r$  ومن  $r$  إلى  $t$ . نختار للمسألة الجزئية الأولى المسار:  $r \rightarrow t \rightarrow s \rightarrow q$ ، وبذلك نكون قد استعملنا أيضًا العقدتين  $s$  و  $t$ . ولن يكون بإمكاننا استعمالهما بعد الآن في حل المسألة الجزئية الثانية، لأن ضم الحلين للمسائلين الجزئيتين سيعطي مسارًا غير بسيط. فإذا تعذَّر علينا استعمال العقدة  $r$  في المسألة الثانية، عندها لن يكون بإمكاننا حل للمسألة على الإطلاق، إذ يتطلب الأمر أن تكون  $t$  على المسار الذي نوجده، وهي ليست العقدة التي نرغب بها خُلِّيَ المسائلين الجزئيين (لأن عقدة الربط هي  $r$ ). ولأننا استخدمنا العقدتين  $s$  و  $t$  في حل إحدى المسائلين الجزئيين، فإننا لا نستطيع استخدامهما في حل للمسألة الجزئية الأخرى. ولكن علينا استخدام إحدى العقدتين على الأقل لحل للمسألة الجزئية الأخرى، وعلينا استخدامهما معًا لحل المسألة أمثلًا. وهكذا نقول إن هاتين المسائلين الجزئيتين ليستا مستقلتين. وبالنظر إلى ذلك بطريقة أخرى، فإن استخدام الموارد في حل إحدى المسائل الجزئية (هذه الموارد هي العقد) يجعلها غير متاحة للمسألة الجزئية الأخرى.

لماذا إذن تكون المسائل الجزئية مستقلة بعضها عن بعض عند إيجاد أقصر مسار؟ الجواب هو أن هذه المسائل الجزئية، بطبيعتها، لا تتشارك في الموارد. إننا ندَّعي أنه لو وجدت عقدة  $w$  على أقصر مسار  $p$  من  $u$  إلى  $v$ ، نستطيع عندها أن نربط بين أي أقصر مسار  $u \xrightarrow{p_1} w$  وأقصر مسار  $w \xrightarrow{p_2} v$  لتوليد أقصر مسار من  $u$  إلى  $v$ . إننا متأكدون من أنه لن تظهر عقدة أخرى غير  $w$  في كلا المسارين  $p_1$  و  $p_2$ . لماذا؟ لنفترض أن عقدة ما  $w \neq x$  تظهر في كلا المسارين  $p_1$  و  $p_2$  بحيث يمكننا تحليل  $p_1$  إلى  $u \xrightarrow{p_{ux}} w$  و  $p_2$  إلى  $w \xrightarrow{p_{wx}} v$ . في البنية الجزئية المثلى لهذه المسألة، يكون للمسار  $p$  وصلات بعدد وصلات  $p_1$  و  $p_2$  معًا، وليكن عدد وصلات  $p$  مساويًا  $e$ . ننشئ الآن المسار  $u \xrightarrow{p_{ux}} x \xrightarrow{p_{xv}} v$  من  $u$  إلى  $v$ . ولأننا أزلنا المسارين من  $x$  إلى  $w$ ، ومن  $w$  إلى  $x$ ، ولكل منهما وصلة على الأقل، فإن المسار  $p'$  يتضمن  $e - 2$  وصلة على الأكثر، وهذا يناقض الفرض بأن  $p$  هو أقصر مسار. وبذلك، نحن متأكدون أن المسائل الجزئية لمسألة أقصر مسار هي مسائل مستقلة.

إن المسألتين اللتين درسناهما في المقطعين 1.15 و 2.15 لهما مسائل جزئية مستقلة. وفي حالة جداء سلسلة مصفوفات، تتمثل للمسائل الجزئية في ضرب سلاسل جزئية  $A_1 A_{i+1} \dots A_k$  و  $A_{k+1} A_{k+2} \dots A_j$ . وفي مسألة تقطيع القضبان، لتحديد أفضل طريق لتقطيع قضيب طوله  $n$ ، ننظر في أفضل طرق تقطيع قضبان أطوالها  $i$  للقيم  $i = 0, 1, \dots, n-1$ . ولما كان الحل الأمثل للمسألة ذات الطول  $n$  يتضمن واحدًا فقط من حلول هذه المسائل الجزئية (بعد أن نكون قد قطعنا أول قطعة)، فلن يكون استقلال الحلول الجزئية نقطة خلاف.

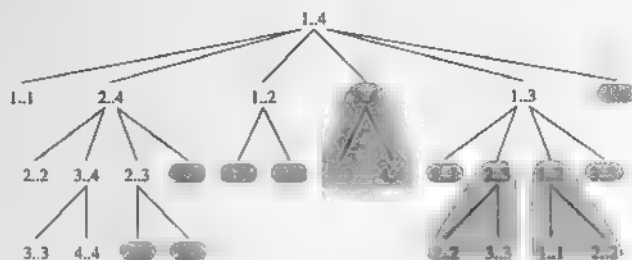
### مسائل جزئية متراكبة

إن المكون الثاني الذي يجب أن تتضمنه مسألة الأمثلة لتكون البرمجة الديناميكية قابلة للتطبيق في حلها هو أن فضاء المسائل الجزئية يجب أن يكون "صغيرًا"، بمعنى أن الخوارزمية العودية التي تحل المسألة تحل المسائل الجزئية نفسها عدة مرات عوضًا عن توليد مسائل جزئية جديدة باستمرار. إن العدد الكلي للمسائل الجزئية المتمايزة هو، نموذجيًا، كثير حدود في حجم المدخل. وحين تعود خوارزمية عودية إلى المسألة نفسها مرات عديدة في الزمن نقول إن لمسألة الأمثلة **مسائل جزئية متراكبة** *overlapping subproblems*. وبالمقابل، فإن المسألة التي يناسبها نهج "فرق-تشد" تولّد عادةً مسائل جديدة عند كل خطوة من العودية. وتستفيد خوارزميات البرمجة الديناميكية عادةً من تراكب للمسائل الجزئية بحل كل مسألة جزئية مرة واحدة، ثم تخزين الحل في جدول يمكن البحث عنه عند الحاجة، باستخدام زمن ثابت للبحث في الجدول.

في المقطع 1.15، درسنا باختصار كيف يُحدّث حلٌّ عؤديٌّ لمسألة تقطيع القضبان استدعاءات كثيرة العدد أسياً لإيجاد حلول لمسائل جزئية أصغر. يخفّض حلُّنا بالبرمجة الديناميكية الخوارزمية العودية بزمنٍ أسّي إلى زمن تربيعي.

ولبيان خاصية المسائل الجزئية المتراكبة بتفصيل أكبر، ندرس ثانية مسألة جداء سلسلة المصفوفات. بالعودة ثانية إلى الشكل 5.15، لاحظ أن الإجراءية MATRIX-CHAIN-ORDER تبحث تكرارًا عن حل للمسائل الجزئية في السطور الدنيا عند البحث عن حل مسائل جزئية في السطور العليا؛ فهي على سبيل المثال تشير إلى العنصر  $m[3,4]$  أربع مرات: أثناء حساب  $m[2,4]$  و  $m[1,4]$  و  $m[3,5]$  و  $m[3,6]$ . ولو أعدنا حساب  $m[3,4]$  في كل مرة عوضًا عن البحث عنها، لازداد زمن التنفيذ زيادةً مثيرة. ولكي نرى كيف

<sup>4</sup> قد يبدو غريبًا أن تعتمد البرمجة الديناميكية على كون المسائل الجزئية مستقلة ومتراكبة. ومع أن هذه المتطلبات قد تبدو متناقضة، فهي توصف مصطلحين مختلفين، لا تقطن على المحور نفسه. نقول عن مسألتين جزئيتين للمسألة نفسها إنهما مستقلتان إذا لم تشكرا ببلوارد. ونقول إنهما متراكبتان إذا كانت هذه للمسائل الجزئية هي فعلاً للمسائل الجزئية ذاتها التي نصادفها كمسائل جزئية لمسائل مختلفة.



الشكل 7.15 شجرة العودية لحساب  $\text{RECURSIVE-MATRIX-CHAIN}(p, 1, 4)$ . تتضمن كل عقدة للموسطات  $i$  و  $j$ . نستعيض عن الحسابات التي تُنجز في الشجرة الفرعية المظلمة بالبحث مرة واحدة فقط في  $\text{MEMOIZED-MATRIX-CHAIN}$ .

يحدث ذلك، ننظر في الإجرائية العودية التالية (غير الفعالة) التي تحدد  $m[i, j]$  وهو أدنى عدد لعمليات الجداء السلمي الضرورية لحساب جداء سلسلة للصفوفات  $A_1 A_{1+1} \dots A_j$ . وتعتمد الخوارزمية مباشرة على العلاقة التكرارية (7.15).

$\text{RECURSIVE-MATRIX-CHAIN}(p, i, j)$

```

1  if  $i == j$ 
2      return 0
3   $m[i, j] = \infty$ 
4  for  $k = i$  to  $j - 1$ 
5       $q = \text{RECURSIVE-MATRIX-CHAIN}(p, i, k)$ 
        +  $\text{RECURSIVE-MATRIX-CHAIN}(p, k + 1, j)$ 
        +  $p_{i-1} p_k p_j$ 
6      if  $q < m[i, j]$ 
7           $m[i, j] = q$ 
8  return  $m[i, j]$ 
```

يبين الشكل 7.15 شجرة العودية التي ينتهجها استدعاء الخوارزمية  $\text{RECURSIVE-MATRIX-CHAIN}(p, 1, 4)$ . جري وضع لصيقة على كل عقدة بقيم للموسطات  $i$  و  $j$ . لاحظ أن بعض أزواج القيم تُرد عدة مرات. والحقيقة أن بإمكاننا إثبات أن الزمن اللازم لحساب  $m[1, n]$  بواسطة هذه الإجرائية العودية هو على الأقل زمن أسّي في  $n$ . نُشير بـ  $T(n)$  إلى الزمن الذي تستغرقه الإجرائية  $\text{RECURSIVE-MATRIX-CHAIN}$  لتحديد وضع الأقواس الأمثل لسلسلة من  $n$  مصفوفة. ولما كان تنفيذ السطور 1-2 والسطور 6-7 يستغرق الواحد منها وحدة زمنية واحدة على الأقل، وكذلك يستغرق الضرب في السطر 5، فإن فحص الإجرائية يعطي التالي:



$$T(1) \geq 1 ,$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \quad \text{for } n > 1 .$$

لاحظ أن كل حد  $T(i)$  (حيث  $i = 1, 2, \dots, n-1$ ) يظهر مرة واحدة في  $T(k)$  ومرة واحدة في  $T(n-k)$ ، وبجميع الوجدان في الجمع  $(n-1)$  مرة مع 1 في المقدمة خارج المجموع، يمكننا إعادة كتابة العلاقة التكرارية السابقة كما يلي:

$$T(n) \geq 2 \sum_{i=1}^{n-1} T(i) + n . \quad (8.15)$$

سنثبت أن  $T(n) = \Omega(2^n)$  بطريقة التعويض. وسنبين، بوجه خاص، أن  $T(n) \geq 2^{n-1}$  لكل قيم  $n \geq 1$ . والقاعدة سهلة، لأن  $T(1) \geq 1 = 2^0$ . وبلاستقراء عندما يكون  $n \geq 2$  لدينا:

$$\begin{aligned} T(n) &\geq 2 \sum_{i=1}^{n-1} 2^{i-1} + n \\ &= 2 \sum_{i=0}^{n-2} 2^i + n \\ &= 2(2^{n-1} - 1) + n \quad (\text{من المعادلة (5.أ)}) \\ &= 2^n - 2 + n \\ &\geq 2^{n-1} \end{aligned}$$

وهذا ينتم البرهان. وبذلك، يكون مقدار العمل الكلي لتخزين بالاستدعاء RECURSIVE-MATRIX-CHAIN(p, 1, n) أسياً في  $n$  على الأقل.

قارن هذه الخوارزمية العودية التزولية بخوارزمية البرمجة الديناميكية الصعودية تجد أن الأخيرة أكثر فعالية لأنها تستفيد من خاصية تراكب المسائل الجزئية، وأن لمسألة جداء المصفوفات  $\Theta(n^2)$  مسألة جزئية متمايزة فقط، وتُحلّ بخوارزمية البرمجة الديناميكية كل مسألة منها مرة واحدة فقط. من جهة أخرى، فإن على الخوارزمية العودية أن تعيد حل كل مسألة جزئية في كل مرة تظهر فيها في شجرة العودية. وكلما تضمنت شجرة العودية، حل مسألة عودية طبيعية، للمسألة الجزئية نفسها تكرارياً، وكان العدد الكلي للمسائل الجزئية المتمايزة صغيراً، فإن البرمجة الديناميكية تستطيع تحسين الفعالية، ويكون هذا التحسين أحياناً مثيراً.

### إعادة إنشاء حل أمثل

كثيراً ما نخزن خياراتنا لكل مسألة جزئية في جدول، بحيث لا يترتب علينا إعادة إنشاء هذه للمعلومات من التكاليف التي خزناها.

وفي مسألة جداء سلسلة المصفوفات، فإن الجدول  $s[i, j]$  يختصر علينا مقدارًا ملحوظًا من العمل حين نعيد إنشاء الحل الأمثل. افترض أننا لم نحفظ بالجدول  $s[i, j]$ ، وأنا ملأنا فقط الجدول  $m[i, j]$  الذي يحتوي تكاليف المسائل الجزئية التلي. نختار من بين الـ  $j - i$  احتمالاً عند تحديد للسائل الجزئية التي علينا أن نستعملها في الحل الأمثل لوضع الأقواس للجداء  $A_1 A_{i+1} \dots A_j$ ، و  $j - i$  ليس ثابتًا. لذلك، سنستغرق وقتًا  $\Theta(j - i) = \omega(1)$  لإعادة إنشاء المسائل الجزئية التي وقع عليها اختيارنا حلاً لمسألة معطاة. ونخزن دليل المصفوفة التي نفرق عندها الجداء  $A_1 A_{i+1} \dots A_j$  في الجدول  $s[i, j]$ ، يمكننا إعادة إنشاء كل الخيارات في زمن  $O(1)$ .

### الاستدكار

مثلما رأينا في مسألة تقطيع القضبان، ثمة نهج بديل عن البرمجة الديناميكية غالبًا ما يوفر فعالية النهج الصعودي للبرمجة الديناميكية مع الاحتفاظ باستراتيجية نزولية. تكمن الفكرة في استدكار *memoize* الخوارزمية العودية الطبيعية غير الفعالة. ومثلما هو الحال في النهج الصعودي، فإننا نحفظ بجدول يتضمن حلول المسائل الجزئية، إلا أن بنية التحكم ملء الجدول هي أشبه بالخوارزمية العودية.

تحتفظ خوارزمية الاستدكار العودية بعنصر في جدول لحل كل مسألة جزئية. يتضمن كل عنصر للجدول مبدئيًا قيمة خاصة تشير إلى أن علينا ملء هذا العنصر. وحين تصادف المسألة الجزئية أول مرة، أثناء نشر الخوارزمية العودية، فإننا نحسب حلها ثم نخزنه في الجدول. وفي كل مرة نتعرض فيها لاحقًا لتلك المسألة الجزئية، فإننا ببساطة نبحث عن قيمتها المخزنة في الجدول ونعيدها إلى الخوارزمية.<sup>5</sup>

وفيما يلي نسخة مستدكر من RECURSIVE-MATRIX-CHAIN. لاحظ مواضع الشبه مع الطريقة المستدكرة النزولية لمسألة تقطيع القضبان.

#### MEMOIZED-MATRIX-CHAIN(p)

```

1   $n = p.length - 1$ 
2  let  $m[1..n, 1..n]$  be a new table
3  for  $i = 1$  to  $n$ 
4      for  $j = i$  to  $n$ 
5           $m[i, j] = \infty$ 
6  return LOOKUP-CHAIN( $m, p, 1, n$ )
```

<sup>5</sup> يفترض هذا النهج سلفًا أننا نعلم مجموعة كل موسطات للمسائل الجزئية للممكنة، وأن العلاقة بين مواقع الجدول والمسائل الجزئية موطدة ومعروفة. وثمة صبح آخر، أكثر عمومية، وهو استدكار القيم باستخدام التهشير (دالة البصمة) مع موسطات المسائل الجزئية كمفاتيح.

LOOKUP-CHAIN( $m, p, i, j$ )

```

1  if  $m[i, j] < \infty$ 
2    return  $m[i, j]$ 
3  if  $i == j$ 
4     $m[i, j] = 0$ 
5  else for  $k = i$  to  $j - 1$ 
6     $q = \text{LOOKUP-CHAIN}(m, p, i, k)$ 
      +  $\text{LOOKUP-CHAIN}(m, p, k + 1, j) + p_{i-1}p_kp_j$ 
7    if  $q < m[i, j]$ 
8       $m[i, j] = q$ 
9  return  $m[i, j]$ 

```

يحتفظ الإجراء MEMOIZED-MATRIX-CHAIN بشأن الإجراء MATRIX-CHAIN-ORDER، بجدول  $m[1..n, 1..n]$  للقيم المحسوبة لـ  $m[i, j]$ ، وهو أصغر عدد للحداءات السلمية اللازمة لحساب المصفوفة  $A_{i..j}$ . يتضمن كل عنصر للجدول مبدئيًا القيمة  $\infty$  لتشير إلى أن علينا ملء العنصر. فإذا جرى استدعاء LOOKUP-CHAIN( $m, p, i, j$ )، وكانت قيمة  $m[i, j]$  أصغر من  $\infty$  في السطر 1، يعيد الإجراء ببساطة التكلفة المحسوبة سابقًا  $m[i, j]$  (في السطر 2)، وإلا، يجري حساب التكلفة من الإجراء RECURSIVE-MATRIX-CHAIN وتخزن في  $m[i, j]$  وتُعاد هذه التكلفة. وهكذا يعيد الإجراء LOOKUP-CHAIN( $m, p, i, j$ ) دائمًا القيمة  $m[i, j]$ ، ولكنه يحسبها فقط في المرة الأولى التي يُستدعى فيها LOOKUP-CHAIN مع هاتين القيمتين المعيّنتين لـ  $i$  و  $j$ .

يبيّن الشكل 7-15 كيف توفر الخوارزمية MEMOIZED-MATRIX-CHAIN الزمن مقارنةً بالخوارزمية RECURSIVE-MATRIX-CHAIN. تمثل الشجرات الجزئية المظللة القيم التي نبحث عنها في الجدول بدلاً من حسابها.

وكما هو الحال في خوارزمية البرمجة الديناميكية MATRIX-CHAIN-ORDER الصعودية، فإن الإجراء MEMOIZED-MATRIX-CHAIN ينفذ في زمن  $O(n^3)$ . ويُنفذ السطر 5 في MEMOIZED-MATRIX-CHAIN بزمن  $\Theta(n^2)$ . يمكننا تصنيف استدعاءات LOOKUP-CHAIN في نوعين:

1. استدعاء يكون فيه  $m[i, j] = \infty$ ، فنُنفذ السطور 3-9 و
2. استدعاء يكون فيه  $m[i, j] < \infty$ ، بحيث تعيد الخوارزمية LOOKUP-CHAIN ببساطة القيمة في السطر 2.

ثمّة  $\Theta(n^2)$  استدعاءً من النوع الأول؛ واحد لكل عنصر في الجدول. تجري جميع الاستدعاءات من النوع الثاني باعتبارها استدعاءات عودية لاستدعاءات من النوع الأول. وكلما أُجرت الخوارزمية LOOKUP-CHAIN استدعاءً عودية، فهي تُجري  $O(n)$  استدعاءً منها. لذلك ثمّة ما مجموعه  $O(n^3)$  استدعاءً من النوع الثاني.

وكل استدعاء من النوع الثاني يستغرق  $O(1)$  من الزمن، في حين يستغرق كل استدعاء من النوع الأول زمناً  $O(n)$  مضافاً إليه الزمن المستغرق في استدعاءاته العودية. فيكون إجمالي الزمن إذن  $O(n^3)$ . وبذلك نحول الاستدكار خوارزمية زمنها  $\Omega(2^n)$  إلى خوارزمية زمنها  $O(n^3)$ .

وبخلاصة القول، يمكن حل مسألة جداء سلسلة مصفوفات إما بخوارزمية استدكار نزولية وإما بخوارزمية برجة ديناميكية صعودية بزمن  $O(n^3)$ . وتستفيد كلتا الطريقتين من خاصية تراكب المسائل الجزئية. يوجد بالمجموع  $\Theta(n^2)$  مسألة جزئية متميزة فقط. ونحسب كل من الطريقتين الحل لكل مسألة جزئية مرة واحدة فقط. وبدون الاستدكار، نُنفذ الخوارزمية العودية الطبيعية بزمن أسّي، لأن المسائل الجزئية يُعاد حلها مرة بعد مرة (تكرارياً).

وبصفة عامة، إذا كان علينا حل جميع المسائل الجزئية مرة واحدة على الأقل، فإن خوارزمية البرجة الديناميكية الصعودية تتفوّق عادةً على خوارزمية الاستدكار النزولية بعامل ثابت، إذ لا يوجد للخوارزمية الصعودية عبء إضافي للعودية، وعبء الاحتفاظ بالجدول هنا أقل من حالة الاستدكار. زدّ على ذلك أن ثمة مسائل يمكن معها الاستفادة من النموذج النظامي للنفاذ إلى الجدول في خوارزمية البرجة الديناميكية تخفيض متطلبات الزمن أو الفضاء (الذاكرة) أكثر فاكثراً. وبدلاً من ذلك، إذا لم يكن علينا حل بعض المسائل الجزئية في فضاء المسائل الجزئية على الإطلاق، فإن طريقة الاستدكار أفضل لأنها تحل فقط للمسائل الجزئية المطلوبة حتماً.

## تمارين

### 1-3.15

أي الطريقتين أكثر فعالية لتحديد العدد الأمثل للجداءات في مسألة جداء سلسلة مصفوفات: غُد كل طرق وضع أقواس الجداء وحساب عدد الجداءات لكل منها، أم تنفيذ خوارزمية RECURSIVE-MATRIX-CHAIN؟ علّل إجابتك.

### 2-3.15

ارسم شجرة العودية للإجرائية MERGE-SORT من للقطع 1-3.2 لصيغة من 16 عنصراً. بيّن لماذا لا يكون الاستدكار فعالاً في تسريع خوارزمية فرز-تسد جيدة مثل MERGE-SORT.

### 3-3.15

ادرس نموذجاً لمسألة جداء سلسلة مصفوفات، الفرض منها وضع الأقواس لمتتالية مصفوفات لجعل عدد الجداءات السلمية اللازمة أعظمياً عوضاً لا أصغرياً. هل تبدي هذه للمسألة بنية جزئية مثلّي؟

### 4-3.15

ذكرنا سابقاً أثناء، في البرجة الديناميكية، تحلّ أولاً للمسائل الجزئية، ثم نختار منها المسائل التي يجب استخدامها في الحل الأمثل للمسألة. ترى الأستاذة Capulet أنه ليس من الضروري دائماً حل جميع المسائل الجزئية لإيجاد

الحل الأمثل. ونقترح أنه يمكن إيجاد الحل الأمثل لمسألة جداء سلسلة مصفوفات دائماً باختيار المصفوفة  $A_k$  التي يجب عليها تفريق الجداء الجزئي  $A_1 A_{i+1} \dots A_j$  (باختيار  $k$  التي تجعل المقدار  $p_{i-1} p_k p_j$  أصغر) قبل حل المسائل الجزئية. أوجد مثلاً على مسألة جداء سلسلة مصفوفات بحيث يكون هذا النهج الشرح حلاً أمثل جزئياً.

### 5-3.15

افترض أننا في مسألة تقطيع القضبان في المقطع 1.15، لدينا أيضاً القيد  $l_i$  على عدد القطع التي طولها  $l_i$  التي يُسمح لنا بإنتاجها، حيث  $i = 1, 2, \dots, n$ . بَيِّنْ أن خاصية البنية الجزئية المثلى الموصفة في المقطع 1.15 لم تعد محققة.

### 6-3.15

تخيل أنك تريد تبديل عملة نقدية. إنك تدرك أنه عوضاً عن تبديل عملة مباشرة بأخرى قد يكون من الأفضل أن تجري سلسلة من التبادلات التجارية باستخدام عملات أخرى، بحيث تنتهي بالعملة التي تريد. افترض أنه يمكنك المتاجرة بـ  $n$  عملة مختلفة مرقمة  $1, 2, \dots, n$ ، حيث تبدأ بالعملة 1 وتنتهي بالعملة  $n$ . يُعطى سعر الصرف  $r_{ij}$  لكل زوج  $i$  و  $j$  من العملات، وهذا يعني أنك إذا بدأت بـ  $d$  وحدة من العملة  $i$ ، فإنك ستبادلها بـ  $d r_{ij}$  وحدة من العملة  $j$ . قد تستلزم عملية الصرف عمولة تعتمد على عدد مرات الصرف التي تجريها. ليكن  $c_k$  مقدار العمولة التي تحثّلها حين تجري  $k$  عملية صرف. بَيِّنْ أنه إذا كان  $c_k = 0$  للقيم  $k = 1, 2, \dots, n$ ، فإن مسألة إيجاد أفضل متتالية للتبادلات من العملة 1 إلى العملة  $n$  تُظهر بنية جزئية مثلى. ثم بَيِّنْ أنه إذا كانت العملات  $c_k$  قيماً اعتباطية، فإن مسألة إيجاد أفضل متتالية للتبادلات من العملة 1 إلى العملة  $n$  لا تُظهر بالضرورة بنية جزئية مثلى.

## 4.15 أطول متتالية جزئية مشتركة

كثيراً ما تحتاج التطبيقات البيولوجية مقارنة سلسلي DNA لكائنين مختلفين (أو أكثر). تتكون جدلية DNA من متتالية جزئيات تسمى **الأسس bases**، حيث الأسس الممكنة هي: الأدينين والغوانين والسيتوزين والثايمين. فإذا مثلنا كلاً من هذه الأسس بالحروف الأول من اسمها اللاتيني، أمكننا التعبير عن جدلية DNA بمتتالية محارف من المجموعة المنتهية  $\{A, C, G, T\}$  (انظر الملحق-ت لتعريف متتالية محارف). على سبيل المثال، يمكن أن تكون جدلية الـ DNA لأحد الكائنات الحية  $S_1 = \text{ACCGGTTCGAGTGCAGCGGAAGCCGGCCGAA}$ ، في حين يمكن أن تكون لكائن حي آخر  $S_2 = \text{GTCGTTCGGAATGCCGTTGCTCTGTAA}$ . ومن دواعي مقارنة جدليتي DNA معرفة مدى "الشابه" بينهما، باعتباره قياساً لمدى قرابة هذين الكائنين. يمكننا تعريف التشابه، ونعرفه فعلاً، بعدة طرق

مختلفة. على سبيل المثال، يمكن أن نقول عن جديلي DNA إنهما متشابهتان إذا كانت إحداها متتالية محارف جزئية من الأخرى. (يناقش الفصل 32 خوارزميات لحل هذه المسألة.) في مثالنا ليست  $S_2$  ولا  $S_1$  متتالية محارف جزئية من الأخرى. بل يمكننا القول عن جديلتين إنهما متشابهتان إذا كان عدد التغيرات اللازمة لتحويل إحدى الجديلتين إلى الأخرى صغيراً (تبحث المسألة 5-15 في هذا المفهوم). وثمة طريقة أخرى لقياس التشابه بين جديلتين  $S_1$  و  $S_2$  تتمثل بإيجاد جدولة ثالثة  $S_3$  تظهر أسسها في كل من  $S_1$  و  $S_2$ ؛ ويجب أن تظهر هذه الأسس بالترتيب نفسه، ولكن ليس بالضرورة على التوالي. وكلما كانت الجدولة  $S_3$  أطول كان التشابه بين  $S_1$  و  $S_2$  أكبر. وفي مثالنا، أطول متتالية  $S_3$  هي GTGTCGGAAGCCGGCCGAA.

يمكن صوغ هذا المفهوم الأخر للتشابه اصطلاحاً على أنه مسألة أطول متتالية جزئية مشتركة. إن متتالية جزئية من متتالية ممطاة هي المتتالية للمطاة نفسها وقد أُشِطَّ منها صفر أو أكثر من عناصرها. فإذا كان لدينا المتتالية  $X = \langle x_1, x_2, \dots, x_m \rangle$ ، فإن متتالية أخرى  $Z = \langle z_1, z_2, \dots, z_k \rangle$  تكون متتالية جزئية *subsequence* من  $X$  إذا وُجدت متتالية متزايدة تماماً  $\langle i_1, i_2, \dots, i_k \rangle$  من أدلة  $X$  بحيث أن لكل قيمة  $z_i = x_{i_i}$ ،  $i = 1, 2, \dots, k$ . على سبيل المثال، إن  $Z = \langle B, C, D, B \rangle$  هي متتالية جزئية من  $X = \langle A, B, C, B, D, A, B \rangle$  والأدلة الموافقة هي  $(2, 3, 5, 7)$ .

ليكن لدينا متتاليتان  $X$  و  $Y$ ، نقول عن متتالية  $Z$  إنها متتالية جزئية مشتركة *common subsequence* لـ  $X$  و  $Y$  إذا كانت  $Z$  متتالية جزئية من كل من  $X$  و  $Y$ . مثلاً، إذا كانت  $X = \langle A, B, C, B, D, A, B \rangle$  و  $Y = \langle B, D, C, A, B, A \rangle$ ، فإن المتتالية  $\langle B, C, A \rangle$  متتالية جزئية مشتركة بين  $X$  و  $Y$ . ولكن المتتالية  $\langle B, C, A \rangle$  ليست أطول متتالية جزئية مشتركة (LCS)، فطول هذه المتتالية 3، وطول المتتالية  $\langle B, C, B, A \rangle$ ، التي هي مشتركة أيضاً بين  $X$  و  $Y$ ، هو 4. إن المتتالية  $\langle B, C, B, A \rangle$  هي إحدى أطول المتتاليات المشتركة لـ  $X$  و  $Y$ ، وكذلك الحال للمتتالية  $\langle B, D, A, B \rangle$ ، إذ ليس لـ  $X$  و  $Y$  أي متتالية جزئية مشتركة بطول 5 أو أكثر.

في مسألة أطول متتالية جزئية مشتركة (LCS) *longest-common-subsequence problem*، لدينا متتاليتان  $X = \langle x_1, x_2, \dots, x_m \rangle$  و  $Y = \langle y_1, y_2, \dots, y_n \rangle$  ونود إيجاد متتالية جزئية مشتركة بين  $X$  و  $Y$  بطول أعظمي. يبيّن هذا للقطع أنه يمكن حل هذه المسألة بفعالية باستخدام البرمجة الديناميكية.

### الخطوة 1: توصيف أطول متتالية جزئية مشتركة

يعتمد نصح البحث الشامل في حل مسألة LCS على عدّ كلِّ المتتاليات الجزئية في  $X$  وفحص كلِّ منها لنرى: هل هي متتالية جزئية من  $Y$  أيضاً؟ وتتبع أطول متتالية جزئية نجدها. توافق كلِّ متتالية جزئية من  $X$  بمجموعة جزئية من الأدلة  $\{1, 2, \dots, m\}$  في  $X$ . ولما كان ثمة  $2^m$  متتالية جزئية في  $X$ ، فإن هذا الأسلوب يتطلب زمناً أسياً، ويجعله غير عملي للمتتاليات الطويلة.

ولكن للمسألة LCS خاصية البنية الجزئية للنسبة، كما تبين المرهنة التالية. وسنرى لاحقاً أن الصفوف الطبيعية للمسائل الجزئية توافق أزواجاً من "السوابق prefixes" متتاليّة الدحل. ولكي نتوخى الدقة، إذا كانت لدينا المتتالية  $X = \langle x_1, x_2, \dots, x_m \rangle$ ، فإننا نعرف السابقة  $i$  للـ  $prefix$  للمتتالية  $X$  لقيم  $i = 0, 1, \dots, m$  على النحو التالي:  $X_i = \langle x_1, x_2, \dots, x_i \rangle$ . فمثلاً، إذا كانت  $X = \langle A, B, C, B, D, A, B \rangle$ ، فإن  $X_4 = \langle A, B, C, B \rangle$  و  $X_0$  هي للمتتالية الخالية.

### مرهنة 1-15 (البنية الجزئية المثلى لمسألة LCS)

لتكن  $X = \langle x_1, x_2, \dots, x_m \rangle$  و  $Y = \langle y_1, y_2, \dots, y_n \rangle$  متتاليتان، ولتكن  $Z = \langle z_1, z_2, \dots, z_k \rangle$  أي متتالية  $LCS$  لـ  $X$  و  $Y$ .

1. إذا كانت  $x_m = y_n$ ، فإن  $z_k = x_m = y_n$  و  $Z_{k-1}$  هي  $LCS$  لـ  $X_{m-1}$  و  $Y_{n-1}$ .
2. إذا كانت  $x_m \neq y_n$ ، فإن  $z_k \neq x_m$  تقتضي أن  $Z$  هي  $LCS$  لـ  $X_{m-1}$  و  $Y$ .
3. إذا كانت  $x_m \neq y_n$ ، فإن  $z_k \neq y_n$  تقتضي أن  $Z$  هي  $LCS$  لـ  $X$  و  $Y_{n-1}$ .

**البرهان (1)** إذا كان  $x_m \neq z_k$  يمكننا أن نلحق  $z_k = y_n = x_m$  للحصول على متتالية جزئية مشتركة لـ  $X$  و  $Y$  بطول  $k+1$ ، وهذا يناقض الفرض أن  $Z$  أطول متتالية جزئية مشتركة لـ  $X$  و  $Y$ . لذلك يجب أن يكون  $z_k = x_m = y_n$ . ثم إن السابقة  $Z_{k-1}$  هي متتالية جزئية مشتركة بين  $X_{m-1}$  و  $Y_{n-1}$  بطول  $k-1$ . نود أن نثبت أن  $LCS$  افترض -مهدف نقض الفرض- أن ثمة متتالية جزئية مشتركة بين  $X_{m-1}$  و  $Y_{n-1}$  بطول أكبر من  $k-1$ ، ولكن هذه المتتالية  $W$  بعد ذلك، نلحق  $z_k = y_n = x_m$  لإنتاج متتالية جزئية مشتركة بين  $X$  و  $Y$ ، طولها أكبر من  $k$ ، وهذا تناقض.

(2) إذا كان  $x_m \neq z_k$  فإن  $Z$  متتالية جزئية مشتركة بين  $X_{m-1}$  و  $Y$ . ولو كان ثمة متتالية جزئية  $W$  مشتركة بين  $X_{m-1}$  و  $Y$ ، طولها أكبر من  $k$ ، لكانت  $W$  أيضاً متتالية جزئية مشتركة بين  $X_m$  و  $Y$ ، وهذا يناقض فرضنا بأن  $Z$  هي  $LCS$  لـ  $X$  و  $Y$ .

(3) البرهان مماثل للحالة (2).

إن الطريقة التي تُوصف بها المرهنة 1.15 للمتتاليات الجزئية المشتركة ذات الطول الأعظم تدل على أن  $LCS$  لمتتاليتين تتضمن  $LCS$  لسابقتين للمتتاليتين. إذن فإن لمسألة  $LCS$  خاصية البنية الجزئية المثلى. ويُصنف الحلّ العودي بخاصية تراكم المسائل الجزئية، كما سنرى قريباً.

### الخطوة 2: حل عودي

نستنتج من المرهنة 1.15 أن علينا دراسة مسألة جزئية واحدة أو اثنتين حين البحث عن  $LCS$

		j	0	1	2	3	4	5	6
i	$x_i$	$y_j$	D	A					
0	$x_0$		0	0	0	0	0	0	0
1	A		0	0	0	0	1	1	1
2			0	1	1	1	1	2	2
3			0	1	1	2	2	2	2
4			0	1	1	2	2	3	3
5	D		0	1	2	2	2	3	3
6			0	1	2	2	3	3	4
7	B		0	1	2	2	3	4	4

الشكل 8.15 الجدولان  $b$  و  $c$  تحسبان من LCS-LENGTH على المتتاليتين  $X = (A, B, C, B, D, A, B)$  و  $Y = (B, D, C, A, B, A)$ . يحتوي المربع في السطر  $i$  والعمود  $j$  القيمة  $c[i, j]$  والسهم المناسب للقيمة  $b[i, j]$ . العنصر 4 في  $c[7, 6]$  - الزاوية اليمنى السفلى في الجدول - هو طول  $\text{LCS}(B, C, B, A)$  و  $X$  و  $Y$ . في حالة  $i, j > 0$  يعتمد العنصر  $c[i, j]$  فقط على كون  $x_i = y_j$  وعلى القيم في العناصر  $c[i-1, j]$  و  $c[i, j-1]$  و  $c[i-1, j-1]$  المحسوبة قبل  $c[i, j]$ . لإعادة بناء عناصر LCS، اتبع سهام  $b[i, j]$  من الزاوية اليمنى السفلى؛ المسار مظلّل. كل "↖" على المسار للطلّ يوافق عنصرًا (مميزًا أو مضاء highlighted) يكون فيه  $x_i = y_j$  عضوًا من LCS.

PRINT-LCS( $b, X, i, j$ )

```

1  if  $i == 0$  or  $j == 0$ 
2      return
3  if  $b[i, j] == "↖"$ 
4      PRINT-LCS( $b, X, i-1, j-1$ )
5      print  $x_i$ 
6  elseif  $b[i, j] == "↑"$ 
7      PRINT-LCS( $b, X, i-1, j$ )
8  else PRINT-LCS( $b, X, i, j-1$ )
    
```

يطبع هذا الإجراء، للجدول  $b$  في الشكل 8.15، للمتتالية BCBA. يستغرق الإجراء زمنًا  $O(m+n)$ ، لأنه في كل مرحلة من العودية يجري على الأقل إنقاص  $i$  أو  $j$  واحدًا.

### تحسين الرماز

ما إن تستكمل تطوير خوارزمية، حتى تجد في الأعم الأغلب أنك تستطيع تحسينها من جهة زمن التنفيذ أو فضاء الذاكرة الذي تستخدمه. يمكن لبعض التغييرات أن تبسط الرماز وتحسن عوامل ثابتة، غير أنها لا تؤدي إلى تحسينات الأداء المقارب. ويمكن أن تؤدي تغييرات أخرى إلى اختراعات مقارنة جوهرية في الزمن وفضاء الذاكرة.



ففي الخوارزمية LCS، على سبيل المثال، يمكننا حذف الجدول  $b$  بكامله. ويعتمد العنصر  $c[i, j]$  على ثلاثة عناصر أخرى فقط للجدول  $c$  هي:  $c[i-1, j-1]$  و  $c[i-1, j]$  و  $c[i, j-1]$ . فإذا علمنا قيمة  $c[i, j]$ ، أمكننا تحديد أي القيم الثلاث السابقة قد استُخِرت في حسابها، في زمن  $O(1)$ ، دون تفحص الجدول  $b$ . وبذلك نستطيع إعادة بناء LCS في زمن  $O(m+n)$  باستخدام إحصائية مشابهة لـ PRINT-LCS. (يُطلب إليك في التمرين 4-4.15 أن تعطي شبه الرمز لذلك.) ومع أننا نوفر فضاء ذاكرة  $\Theta(mn)$  بهذه الطريقة، إلا أن متطلبات فضاء الذاكرة للمساعدة لحساب LCS لا تنقص نقصاً مقارباً لأننا على أي حال، نحتاج إلى فضاء  $\Theta(mn)$  للجدول  $c$ .

ومع ذلك، يمكننا خفض متطلبات الفضاء على نحو مقارب لـ LCS-LENGTH لأنها تحتاج فقط إلى سطرين من الجدول  $c$  في كل مرة؛ السطر الذي هو في قيد الحساب والسطر السابق. (في الحقيقة، وكما يُطلب إليك في التمرين 4-4.15 أن تبين أن بالإمكان استخدام فضاء أكبر بقليل من فضاء السطر الواحد من  $c$  لحساب طول LCS.) ويصح هذا التحسين إذا كنا نحتاج إلى طول LCS فقط؛ أما إذا احتجنا إلى إعادة بناء عناصر الـ LCS فإن الجدول الأصغر لا يحتفظ بالمعلومات الكافية لانتفاء أثر خطواتنا في زمن  $O(m+n)$ .

#### تمارين

##### 1-4.15

حدّد LCS لـ  $\{1, 0, 0, 1, 0, 1, 0, 1\}$  و  $\{0, 1, 0, 1, 1, 0, 1, 0\}$ .

##### 2-4.15

أعط شبه الرمز لإعادة بناء LCS من الجدول  $c$  الكامل وللتأليتين الأصليتين  $X = \langle x_1, x_2, \dots, x_m \rangle$  و  $Y = \langle y_1, y_2, \dots, y_n \rangle$  في زمن  $O(m+n)$ ، من دون استخدام الجدول  $b$ .

##### 3-4.15

أعط نسخة مستديرة من LCS-LENGTH تُنفذ بزمن  $O(mn)$ .

##### 4-4.15

بَيّن طريقة حساب طول LCS باستخدام  $2 \times \min(m, n)$  عنصراً في الجدول  $c$  إضافة إلى  $O(1)$  فضاء إضافي. ثم بَيّن كيف يمكنك فعل ذلك باستخدام  $\min(m, n)$  عنصراً و  $O(1)$  فضاء إضافي.

##### 5-4.15

أعط خوارزمية تنفذ بزمن  $O(n^2)$  لإيجاد أطول متتالية جزئية متزايدة باطراد من متتالية مؤلفة من  $n$  عدداً.

##### \* 6-4.15

أعط خوارزمية تنفذ بزمن  $O(n \lg n)$  لإيجاد أطول متتالية جزئية متزايدة باطراد من متتالية مؤلفة من  $n$  عدداً.

(تلميح: لاحظ أن آخر عنصر من متتالية جزئية مرشحة للإجابة بطول  $l$  هو على الأقل بكرر آخر عنصر من متتالية جزئية مرشحة للإجابة بطول  $l-1$ . احتفظ بالمتتاليات الجزئية المرشحة بربطها معًا من خلال متتالية الدخل.)

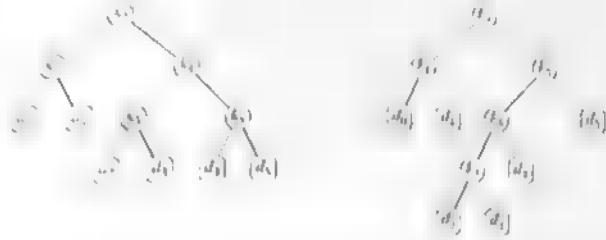
## 5.15 شجرات البحث الثنائية المثلى

لفترض أننا نصمم برنامجًا لترجمة نص من الإنكليزية إلى الفرنسية. إننا نحتاج، عند كل ورود لكل كلمة إنكليزية في النص، إلى البحث عن مكافئها الفرنسي. يمكننا إنجاز عملية البحث هذه في بناء شجرة بحث ثنائية مفاتيحها  $n$  كلمة الإنكليزية، ومعطياتها التابعة هي المكافئات الفرنسية. ولأننا سنبحث في الشجرة عن كل كلمة مفردة في النص، نود أن يكون الزمن الكلي المصروف في البحث أصغر ما يمكن. يمكننا ضمان زمن بحث  $O(\lg n)$  لكل ورود باستخدام شجرة حمراء-سوداء أو أي شجرة بحث ثنائية متوازنة أخرى. على أن الكلمات تظهر بتواترات مختلفة، ويمكن أن تكون حالة الشجرة بحيث تظهر كلمة كثيرة الاستخدام مثل "the" بعيدة عن جذر الشجرة، في حين تظهر كلمة نادرة الاستخدام مثل "machicolation" قريبة من الجذر. ومن شأن مثل هذا التنظيم أن يبطل الترجمة، لأن عدد العقد التي نزورها حين نبحث عن مفتاح في شجرة بحث ثنائية هو واحد مضافًا إلى عمق العقدة التي تتضمن المفتاح. ونحن نود أن نوضح الكلمات العالية الورد في النص قرب الجذر.<sup>6</sup> أضف إلى ذلك إمكان ورود كلمات في النص ليس لها ترجمة فرنسية،<sup>7</sup> ويمكن ألا تظهر هذه الكلمات في شجرة البحث الثنائية على الإطلاق. فكيف يمكننا تنظيم شجرة البحث الثنائية بحيث يكون عدد العقد التي نزورها لجميع عمليات البحث أصغرًا، إذا كنا نعلم تواتر ورود كل كلمة؟

ما نريده يُعرف باسم **شجرة بحث ثنائية مثلى** *optimal binary search tree*. صورًا، لدينا متتالية  $K = \{k_1, k_2, \dots, k_n\}$  من  $n$  مفتاحًا متميزًا، بترتيب مغزول (أي أن  $k_1 < k_2 < \dots < k_n$ )، ونود بناء شجرة بحث ثنائية من هذه المفاتيح. لدينا لكل مفتاح  $k_i$  الاحتمال  $p_i$  وهو احتمال أن يكون البحث عن  $k_i$  يمكن لبعض عمليات البحث أن تكون لقيم ليست في  $K$ ، لذلك لدينا أيضًا  $n+1$  "مفتاحًا شكليًا" هي  $d_0, d_1, d_2, \dots, d_n$  تمثل قيمًا غير موجودة في  $K$ . وعلى وجه الخصوص، تمثل  $d_0$  كل القيم التي هي أقل من  $k_1$ ، وتمثل  $d_n$  كل القيم التي هي أكبر من  $k_n$ . وفي حالة  $i = 1, 2, \dots, n-1$ ، فإن المفتاح الشكلي  $d_i$  يمثل كل القيم التي هي بين  $k_i$  و  $k_{i+1}$ . ولكل مفتاح شكلي  $d_i$  لدينا الاحتمال  $q_i$  أن يتطابق البحث مع  $d_i$ . يبين الشكل 9.15 شجري بحث ثنائيين لمجموعة من خمس مفاتيح،  $n = 5$ . يكون كل مفتاح  $k_i$

<sup>6</sup> إذا كان موضوع النص حول بيان القلاع، قد نود أن تظهر كلمة "machicolation" قرب الجذر.

<sup>7</sup> نعم للكلمة *machicolation* مقابل فرنسي: *mâchioulis*.



الشكل 9.15 شجرة بحث ثنائية لمجموعة من خمسة مفاتيح،  $n = 5$ . هنا الاحتمالات ثابتة:

5	4	3	2	1	0	$i$
0.20	0.10	0.05	0.10	0.15	0.05	$p_i$
0.10	0.05	0.05	0.05	0.10		$q_i$

(أ) شجرة بحث ثنائية بتكلفة بحث متوقعة (وسطى) 2.80. (ب) شجرة بحث ثنائية بتكلفة بحث متوقعة 2.75. هذه الشجرة مثلى.

عقدة داخلية، وكل مفتاح شكلي  $k_i$  ورقة. ويكون كل بحث إما ناجحاً (يُوجد مفتاحاً ما  $k_i$ ) وإما غير ناجح (يُوجد مفتاحاً شكلياً ما  $d_i$ ). ونلاحظ يكون لدينا:

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1. \quad (10.15)$$

ولما كان لدينا احتمالات بحث لكل مفتاح ولكل مفتاح شكلي، فيمكننا تحديد تكلفة متوقعة لبحث ما في شجرة بحث ثنائية بمغارة  $T$ . لنفترض أن التكلفة الفعلية للبحث تساوي عدد العقد التي نتقها، أي عمق العقدة التي نلحظها بالبحث في  $T$  مضافاً له 1. عندها تكون التكلفة المتوقعة لبحث في  $T$  هي:

$$\begin{aligned} E[\text{search cost in } T] &= \sum_{i=1}^n (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^n (\text{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^n \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^n \text{depth}_T(d_i) \cdot q_i, \end{aligned} \quad (11.15)$$

حيث يشير  $\text{depth}_T$  إلى عمق العقدة في الشجرة  $T$ . وللمساواة الأخيرة ننتج بالضرورة من معادلة (10.15). وفي الشكل 9-15أ، يمكننا حساب تكلفة البحث عقدة بعقدة:

عقدة	عمقها	الاحتمال	المساهمة
$k_1$	1	0.15	0.30
$k_2$	0	0.10	0.10
$k_3$	2	0.05	0.15
$k_4$	1	0.10	0.20
$k_5$	2	0.20	0.60
$d_0$	2	0.05	0.15
$d_1$	2	0.10	0.30
$d_2$	3	0.05	0.20
$d_3$	3	0.05	0.20
$d_4$	3	0.05	0.20
$d_5$	3	0.10	0.40
المجموع			2.80

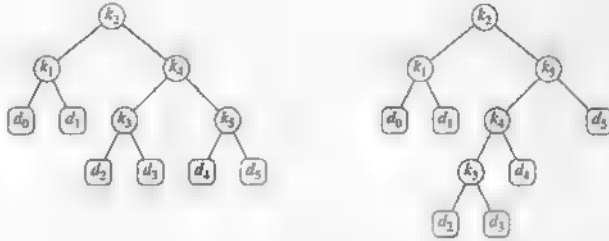
وفي حالة مجموعة معطاة من الاحتمالات، فإن هدفنا هو بناء شجرة بحث ثنائية بحيث تكون تكلفة البحث المتوقعة فيها أصغر ما يمكن. نسمي مثل هذه الشجرة *شجرة بحث ثنائية مثلى* *optimal binary search tree*. ويبين الشكل 9.15 (ب) شجرة بحث ثنائية مثلى للاحتمالات للعطاة في ترويسة الشكل؛ تكلفتها المتوقعة 2.75. يبين هذا المثال أن شجرة البحث الثنائية المثلى ليست بالضرورة الشجرة ذات الارتفاع الكلي الأصغر. وأنه لا يمكننا بالضرورة بناء شجرة البحث الثنائية المثلى بوضع المفتاح ذي الاحتمال الأعلى عند الجذر. فهنا يكون احتمال البحث عن المفتاح  $k_5$  أعلى من احتمال البحث عن أي مفتاح آخر، ومع ذلك فإن جذر شجرة البحث الثنائية المثلى هو  $k_2$ . (إن أدنى تكلفة متوقعة لأي شجرة بحث ثنائية يكون  $k_5$  عند جذرها هي 2.85).

وكما هو الحال في جداء سلسلة المصفوفات، يحقق البحث الشامل لكل الاحتمالات في تحقيق عوارزمية فعالة. وبإمكاننا وضع لصيقة على عُقد أي شجرة ثنائية من  $n$  عقدة تتضمن المفاتيح  $k_1, k_2, \dots, k_n$  لبناء شجرة بحث ثنائية، ثم نضيف المفاتيح الشكلية كورقات. وقد رأينا في المسألة 4-12 أن عدد الشجرات الثنائية التي تتضمن  $n$  عقدة هو  $\Omega(4^n/n^{3/2})$ ، وبذلك يكون علينا فحص عددٍ أُسّي من شجرات البحث الثنائية في البحث الشامل. وليس مستغرباً، أن نحلّ المسألة بالبرمجة الديناميكية.

### الخطوة 1: بنية شجرة بحث ثنائية مثلى

لتوصيف البنية الجزئية المثلى لأشجار البحث الثنائية المثلى، نبدأ بملاحظة تتعلق بالشجرات الفرعية. خذ أي شجرة فرعية من شجرة بحث ثنائية؛ يجب أن تتضمن هذه الشجرة مفاتيح في مجال متصل  $k_i, \dots, k_j$ ، لقيم  $1 \leq i \leq j \leq n$ . إضافة إلى ذلك، فإن الشجرة الفرعية التي تتضمن المفاتيح  $k_i, \dots, k_j$  يجب أن تتضمن أيضاً المفاتيح الشكلية  $d_{i-1}, \dots, d_j$  باعتبارها وريقات لها.

الآن يمكننا التعبير عن البنية الجزئية المثلى: إذا كان لدينا شجرة بحث ثنائية مثلى  $T$  تحتوي شجرة فرعية



الشكل 9.15 شجرنا بحث ثنائيتان لمجموعة من خمسة مفاتيح،  $n = 5$ ، لها الاحتمالات التالية:

5	4	3	2	1	0	$i$
0.20	0.10	0.05	0.10	0.15	0.05	$p_i$
0.10	0.05	0.05	0.05	0.10		$q_i$

(أ) شجرة بحث ثنائية بتكلفة بحث متوقعة (وسطى) 2.80. (ب) شجرة بحث ثنائية بتكلفة بحث متوقعة 2.75. هذه الشجرة مثلى.

عقدة داخلية، وكل مفتاح شكلي  $d_i$  ورقة. ويكون كل بحث إما ناجحاً (يُوجد مفتاحاً ما  $k_i$ ) وإما غير ناجح (يُوجد مفتاحاً شكلياً ما  $d_i$ )، وبذلك يكون لدينا:

$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1. \quad (10.15)$$

ولما كان لدينا احتمالات بحث لكل مفتاح ولكل مفتاح شكلي، فيمكننا تحديد التكلفة المتوقعة لبحث ما في شجرة بحث ثنائية معطاة  $T$ . لنفترض أن التكلفة الفعلية للبحث تساوي عدد العقد التي نطورها، أي عمق العقدة التي نبحثها بالبحث في  $T$  مضافاً له 1. عندها تكون التكلفة المتوقعة للبحث في  $T$  هي:

$$\begin{aligned} E[\text{search cost in } T] &= \sum_{i=1}^n (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^n (\text{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^n \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^n \text{depth}_T(d_i) \cdot q_i, \end{aligned} \quad (11.15)$$

حيث يشير  $\text{depth}_T$  إلى عمق العقدة في الشجرة  $T$ . وللساواة الأخيرة تنتج بالضرورة من المعادلة (10.15). وفي الشكل 9-15، يمكننا حساب تكلفة البحث عقدة بعقدة:

المساهمة	الاحتمال	عمقها	عقدة
0.30	0.15	1	$k_1$
0.10	0.10	III	$k_2$
0.15	0.05	2	$k_3$
0.20	0.10	1	$k_4$
0.60	0.20	2	$k_5$
0.15	0.05	2	$d_0$
0.30	0.10	2	$d_1$
0.20	0.05	3	$d_2$
0.20	0.05	3	$d_3$
0.20	0.05	3	$d_4$
0.40	0.10	III	$d_5$
2.80			المجموع

وفي حالة مجموعة معطاة من الاحتمالات، فإن هدفنا هو بناء شجرة بحث ثنائية بحيث تكون تكلفة البحث المتوقعة فيها أصغر ما يمكن. نسمي مثل هذه الشجرة **شجرة بحث ثنائية مثلى** *optimal binary search tree*. ويبين الشكل 9.15 (ب) شجرة بحث ثنائية مثلى للاحتمالات المعطاة في ترويسة الشكل 1. تكلفتها المتوقعة 2.75. يبين هذا المثال أن شجرة البحث الثنائية المثلى ليست بالضرورة الشجرة ذات الارتفاع الكلي الأصغر. وأنه لا يمكننا بالضرورة بناء شجرة البحث الثنائية المثلى بوضع المفتاح ذي الاحتمال الأعلى عند الجذر. فهنا يكون احتمال البحث عن المفتاح  $k_5$  أعلى من احتمال البحث عن أي مفتاح آخر، ومع ذلك فإن جذر شجرة البحث الثنائية المثلى هو  $k_2$ . (إن أدنى تكلفة متوقعة لأي شجرة بحث ثنائية يكون  $k_5$  عند جذورها هي 2.85).

وكما هو الحال في جداء سلسلة المصفوفات، يحقق البحث الشامل لكل الاحتمالات في تحقيق خوارزمية فعالة. وبإمكاننا وضع لصيقة على عقدة أي شجرة ثنائية من  $n$  عقدة تتضمن المفاتيح  $k_1, k_2, \dots, k_n$  لبناء شجرة بحث ثنائية، ثم نضيف المفاتيح الشكلية كورقات. وقد رأينا في المسألة 4-12 أن عدد الشجرات الثنائية التي تتضمن III عقدة هو  $\Omega(4^n/n^{3/2})$ ، وبذلك يكون علينا فحص عددٍ أُسِّي من شجرات البحث الثنائية في البحث الشامل. وليس مستغرباً، أن نحل المسألة بالبرمجة الديناميكية.

### الخطوة 1: بنية شجرة بحث ثنائية مثلى

لتوصيف البنية الجزئية المثلى لأشجار البحث الثنائية المثلى، نبدأ بملاحظة تتعلق بالشجرات الفرعية. نحذ أي شجرة فرعية من شجرة بحث ثنائية؛ يجب أن تتضمن هذه الشجرة مفاتيح في مجال متصل  $k_i, \dots, k_j$ ، لقيم  $1 \leq i \leq j \leq n$ . إضافة إلى ذلك، فإن الشجرة الفرعية التي تتضمن المفاتيح  $k_i, \dots, k_j$ ، يجب أن تتضمن أيضاً المفاتيح الشكلية  $d_i, \dots, d_{j-1}$  باعتبارها وريقات لها.

الآن يمكننا التعبير عن البنية الجزئية المثلى: إذا كان لدينا شجرة بحث ثنائية مثلى  $T$  تحتوي شجرة فرعية

$T'$  تتضمن المفاتيح  $k_1, \dots, k_r$ ، وجب أن تكون الشجرة الفرعية  $T'$  مثل  $T$  أيضاً للمسائل الجزئية ذات المفاتيح  $k_1, \dots, k_r$  والمفاتيح الشكلية  $d_1, \dots, d_{l-1}$ . وتصح هنا قاعدة "قص والصق" للعدادة. لو كان ثمة شجرة فرعية  $T''$  تكلفتها للتوقعة أقل من التكلفة للتوقعة لـ  $T'$ ، عندها يمكننا قص  $T'$  من  $T$  ولصق  $T''$  مكانها، فنتج شجرة بحث ثنائية بتكلفة متوقعة أقل من التكلفة للتوقعة لـ  $T$ ، وهذا يناقض كون  $T$  مثلي.

نحتاج إلى استخدام بنية جزئية مثلي لتبين أننا نستطيع بناء حل أمثل للمسألة من الحلول المثلي للمسائل الجزئية. فإذا كان لدينا للمفاتيح  $k_1, \dots, k_r$ ، فإن أحد هذه المفاتيح وليكن  $k_r$  حيث  $(1 \leq r \leq l)$ ، سيكون جذر شجرة فرعية مثلي تتضمن هذه المفاتيح. وتحتوي الشجرة الفرعية اليسرى للجذر  $k_r$  المفاتيح  $k_1, \dots, k_{r-1}$  (والمفاتيح الشكلية  $d_1, \dots, d_{r-1}$ )، وتحتوي الشجرة الفرعية اليمنى للمفاتيح  $k_1, \dots, k_{r-1}$  (والمفاتيح الشكلية  $d_1, \dots, d_r$ ). ومادامنا نفحص كل الجذور المرشحة  $k_r$  حيث  $1 \leq r \leq l$ ، ونحدد كل شجرات البحث الثنائية المثلي التي تتضمن للمفاتيح  $k_1, \dots, k_{r-1}$  وتلك التي تتضمن للمفاتيح  $k_1, \dots, k_r$ ، فمن المؤكد أننا سنجد شجرة بحث ثنائية مثلي.

ثمة تفصيل جدير بالملاحظة يتعلق بالشجرات الفرعية "الفارغة". لنفترض أننا، في شجرة فرعية تتضمن المفاتيح  $k_1, \dots, k_r$ ، اخترنا  $k_l$  جذراً. فاستناداً إلى الحجة المبينة آنفاً، فإن الشجرة الفرعية اليسرى التي جذرها  $k_l$  تتضمن المفاتيح  $k_1, \dots, k_{l-1}$ . ومن الطبيعي أن نفر هذه للتالية على أنها لا تتضمن أية مفاتيح. ولكن، تذكر دائماً أن الشجرات الفرعية تتضمن أيضاً مفاتيح شكلية. وسنستخدم اصطلاح أن الشجرة الفرعية التي تتضمن للمفاتيح  $k_1, \dots, k_{l-1}$  لا تتضمن مفاتيح فعلية، لكنها تتضمن المفتاح الشكلي الوحيد  $d_{l-1}$  فقط. وبالتناظر، لو اخترنا  $k_r$  جذراً، فإن الشجرة الفرعية اليمنى التي جذرها  $k_r$  تتضمن للمفاتيح  $k_1, \dots, k_{r-1}$  ولا تتضمن هذه الشجرة الفرعية اليمنى أي مفاتيح فعلية، إلا أنها تتضمن المفتاح الشكلي  $d_r$ .

## الخطوة 2: حل عودي

أصبحنا الآن مهيين لتعريف قيمة الحل الأمثل عودياً. نتناول نطاق مسائلنا الجزئية على أنه إيجاد شجرة بحث ثنائية مثلي تتضمن المفاتيح  $k_1, \dots, k_r$ ، حيث  $1 \leq l$  و  $z \leq n$  و  $1 \leq l-1$  و  $z$ . (لاحظ أنه عندما يكون لدينا  $l-1 = z$  لا يكون ثمة أي مفاتيح فعلية؛ ويكون لدينا فقط المفتاح الشكلي  $d_{l-1}$ ). لنعرّف  $e[l, z]$  على أنها التكلفة للتوقعة للبحث في شجرة بحث ثنائية مثلي تتضمن للمفاتيح  $k_1, \dots, k_r$ . ونود في نهاية المطاف حساب  $e[1, n]$ .

نحدث الحالة السهلة حين يكون  $l-1 = z$ . عندها يكون لدينا فقط المفتاح الشكلي  $d_{l-1}$ ، ونكون تكلفة البحث للتوقعة  $e[l, l-1] = q_{l-1}$ .

وعندما يكون  $l \geq z$ ، نحتاج إلى اختيار جذر  $k_r$  من بين المفاتيح  $k_1, \dots, k_r$  ثم إنشاء شجرة بحث ثنائية مثلي بحيث تكون للمفاتيح  $k_1, \dots, k_{r-1}$  شجرتها الفرعية اليسرى وإنشاء شجرة بحث ثنائية مثلي بحيث تكون

المفاتيح  $k_r, k_{r+1}, \dots, k_r$  شجرة الفرعية اليمنى. ماذا يحصل لتكلفة البحث المتوقعة لشجرة فرعية حين تصبح هذه الشجرة الفرعية شجرة فرعية لعقدة؟ يزداد عمق كل عقدة في الشجرة فرعية بـ 1. ومن العلاقة (11.15) تزداد تكلفة البحث المتوقعة لهذه الشجرة الفرعية بمجموع كل الاحتمالات في الشجرة الفرعية. وفي حالة شجرة فرعية تتضمن المفاتيح  $k_r, k_{r+1}, \dots, k_r$ ، ل نرمز إلى مجموع الاحتمالات هذا بـ

$$w(i, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l \quad (12.15)$$

ومن ثم، إذا كان  $k_r$  جذر شجرة فرعية على  $k_r, k_{r+1}, \dots, k_r$  يكون لدينا

$$e(i, j) = p_r + (e(i, r-1) + w(i, r-1) + (e(r+1, j) + w(r+1, j))) .$$

لاحظ أن:

$$w(i, j) = w(i, r-1) + p_r + w(r+1, j) ,$$

نعيد كتابة  $e(i, j)$  كما يلي

$$e(i, j) = r(i, r-1) + e(r+1, j) + w(i, j) \quad (13.15)$$

نفترض العلاقة العودية (13.15) أننا نعلم أي عقدة  $k_r$  سنختارها جذراً. نختار الجذر الذي يعطي أقل تكلفة بحث متوقعة، وهذا يعطينا الصيغة التكرارية النهائية:

$$e(i, j) = \begin{cases} q_{i-1} & \text{if } j = i-1 , \\ \min_{r=i \dots j} \{e(i, r-1) + e(r+1, j) + w(i, j)\} & \text{if } i \leq j . \end{cases} \quad (14.15)$$

نعطي قيم  $e(i, j)$  تكاليف البحث المتوقعة في شجرات البحث الثنائية المثلى. ولمساعدتنا في اقتفاء أثر بنية شجرات البحث الثنائية المثلى، نعرف  $root[i, j]$  للقيم  $1 \leq i \leq j \leq n$  على أنه الدليل  $r$  حيث  $k_r$  جذر شجرة بحث ثنائية مثلى تتضمن المفاتيح  $k_r, k_{r+1}, \dots, k_r$ . ومع أننا سنرى طريقة حساب قيم  $root[i, j]$ ، سندع بناء شجرة البحث الثنائية المثلى من هذه القيم للتمرين 15.5-I.

### الخطوة 3: حساب تكلفة البحث المتوقعة لشجرة بحث ثنائية مثلى

لذلك لاحظت، عند هذه النقطة، التشابهات بين توصيفنا لشجرات البحث الثنائية المثلى وجداء سلسلة المصفوفات. ففي كلا نطاقَي المسألتين، تتكون مسائلنا الجزئية من مجالات جزئية ذات أدلة متصلة. وسيكون التمييز لباحث العودي للمعادلة (14.15) غير فعال كما كان الحال في الخوارزمية المباشرة العودية لجداء سلسلة مصفوفات. وعوضاً عن ذلك، نخزن قيم  $e(i, j)$  في جدول  $e[1..n+1, 0..n]$ . يحتاج الدليل الأول أن يذهب حتى  $n+1$  عوضاً عن  $n$ ، ذلك أننا لكي نحصل على شجرة فرعية تتضمن المفتاح الشكلي  $d_n$  فقط، نحتاج إلى حساب  $e[n+1, n]$  ونحزنها. وبحاجة الدليل الثاني أن يبدأ من 0 كي نحصل على شجرة فرعية



تتضمن المفتاح الشكلي  $d_0$  فقط، لذلك نحتاج إلى حساب  $e[1,0]$  وعزماً. نستخدم العناصر  $e[i,j]$  التي يكون لها  $i-1 \geq j$  فقط. ونستخدم أيضاً الجدول  $root[i,j]$  لتسجيل جذر الشجرة الفرعية التي تتضمن المفاتيح  $k_1, \dots, k_j$ . يستخدم هذا الجدول العناصر التي تحقق  $1 \leq i \leq n$  فقط.

سنحتاج إلى جدول آخر لزيادة الفعالية. فعوضاً عن حساب  $w(i,j)$  من العدم في كل مرة نحسب فيها  $e[i,j]$  - وهذا يستغرق  $\Theta(j-i)$  عملية جمع- فإننا نخزن هذه القيم في جدول  $w[1..n+1, 0..n]$ . وفي الحالة الأساسية، نحسب  $w[i, i-1] = q_{i-1}$  حين يكون  $1 \leq i \leq n+1$ . أما في حالة  $i \geq j$  فإننا نحسب:

$$w[i,j] = w[i,j-1] + p_j + q_j. \quad (15.15)$$

وهكذا يمكننا حساب  $\Theta(n^2)$  قيمة لـ  $w[i,j]$  بزمن  $\Theta(1)$  لكل منها.

إن دخل شبه الرمز التالي هو: الاحتمالات  $p_1, \dots, p_n$  و  $q_0, \dots, q_n$  والحجم  $n$ . وهو يعيد الجدولين  $\blacksquare$  و  $root$ .

OPTIMAL-BST( $p, q, n$ )

```

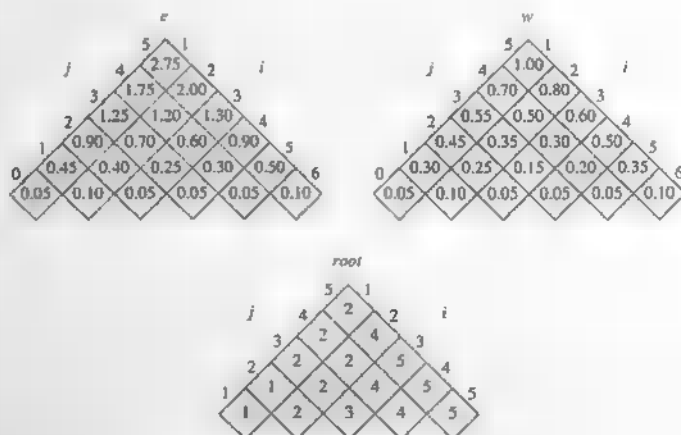
1  let  $e[1..n+1, 0..n]$ ,  $w[1..n+1, 0..n]$ ,
    and  $root[1..n, 1..n]$  be new tables
2  for  $i = 1$  to  $n+1$ 
3       $e[i, i-1] = q_{i-1}$ 
4       $w[i, i-1] = q_{i-1}$ 
5  for  $l = 1$  to  $n$ 
6      for  $i = 1$  to  $n-l+1$ 
7           $j = i+l-1$ 
8           $e[i,j] = \infty$ 
9           $w[i,j] = w[i,j-1] + p_j + q_j$ 
10         for  $r = i$  to  $j$ 
11              $t = e[i, r-1] + e[r+1, j] + w[i,j]$ 
12             if  $t < e[i, j]$ 
13                  $e[i, j] = t$ 
14                  $root[i, j] = r$ 
15  return  $e$  and  $root$ 
```

من هذا الوصف، ومن التشابه مع إجرائية MATRIX-CHAIN-ORDER في اللفظ 2.15، نتجد أن عمل هذه الإجرائية واضح ومباشر. فحلقة for في السطور 4-2 تستبدل قيم  $e[i, i-1]$  و  $w[i, i-1]$ . ثم تستخدم حلقة for في السطور 5-14 العلاقات العودية (14.15) و (15.15) لحساب  $e[i, j]$  و  $w[i, j]$  للقيم  $1 \leq i \leq j \leq n$ . وفي التكرار الأول حين يكون  $l = 1$  نحسب الحلقة  $e[i, i]$  و  $w[i, i]$  لكل  $i = 1, 2, \dots, n$ . وفي التكرار الثاني، حين تكون  $l = 2$  يجري حساب  $e[i, i+1]$  و  $w[i, i+1]$  لكل

$i = 1, 2, \dots, n-1$ ، وهكذا. أما حلقة **for** الأعمق (الأكثر داخلية) في السطور 10-14 فتحزّب كل دليل مرشح  $r$  لتحديد أي مفتاح  $k_r$  يجب استخدامه جذراً لشجرة بحث ثنائية مثلى تتضمن المفاتيح  $k_1, \dots, k_r$ . تحزّب حلقة **for** هذه القيم الحالية للدليل  $r$  في  $root[i, j]$  كلما وجدت مفتاحاً أفضل يمكن استخدامه جذراً.

يبين الشكل 10.15 الجداول  $e[i, j]$  و  $w[i, j]$  و  $root[i, j]$  المحسوبة في الإجرائية OPTIMAL-BST لتوزيع المفاتيح المبين بالشكل 9.15. وكما هو الحال في مثال جداء سلسلة المصفوفات للشكل 5.15، تدوّر الجداول لجعل الأقطار أفقية. تُحسب OPTIMAL-BST السطور من الأسفل إلى الأعلى ومن اليسار إلى اليمين ضمن كل سطر.

تستغرق الإجرائية OPTIMAL-BST زمناً  $\Theta(n^3)$ ، تماماً مثل MATRIX-CHAIN-ORDER. من السهل ملاحظة أن زمن التنفيذ  $O(n^3)$ ، لأن الإجرائية تضم ثلاث حلقات **for** متداخلة، وكل دليل حلقة يأخذ قيماً تساوي  $n$  على الأكثر. وليس لأدلة الحلقات في OPTIMAL-BST الحدود نفسها تماماً الموجودة في MATRIX-CHAIN-ORDER، إلا أنها تختلف على الأكثر بـ 1 في كل الاتجاهات. لذلك، وكما في MATRIX-CHAIN-ORDER، تستغرق الإجرائية OPTIMAL-BST زمناً  $\Omega(n^3)$ .



**الشكل 10.15** الجداول  $e[i, j]$  و  $w[i, j]$  و  $root[i, j]$  محسوبة بالإجرائية OPTIMAL-BST لتوزيع المفاتيح المبين بالشكل 9.15. جرى تدوير الجداول لتبدو الأقطار أفقية.

## تمارين

### 1-5.15

اكتب شبه رماز للإجرائية  $\text{CONSTRUCT-OPTIMAL-BST}(\text{root})$  التي تُخرج بنية شجرة بحث ثنائية مثلى، من جدول  $\text{root}$  معطى. يجب أن تطبع الإجرائية البنية التالية في حالة مثال الشكل 10.15.

$k_2$  is the root  
 $k_1$  is the left child of  $k_2$   
 $d_0$  is the left child of  $k_1$   
 $d_1$  is the right child of  $k_1$   
 $k_5$  is the right child of  $k_2$   
 $k_4$  is the left child of  $k_5$   
 $k_3$  is the left child of  $k_4$   
 $d_2$  is the left child of  $k_3$   
 $d_3$  is the right child of  $k_3$   
 $d_4$  is the right child of  $k_4$   
 $d_5$  is the right child of  $k_5$

والتي تعني على الترتيب:

$k_2$  هي الجذر

$k_1$  الابن الأيسر لـ  $k_2$

$d_0$  الابن الأيسر لـ  $k_1$

$d_1$  الابن الأيمن لـ  $k_1$

$k_5$  الابن الأيمن لـ  $k_2$

$k_4$  الابن الأيسر لـ  $k_5$

$k_3$  الابن الأيسر لـ  $k_4$

$d_2$  الابن الأيسر لـ  $k_3$

$d_3$  الابن الأيمن لـ  $k_3$

$d_4$  الابن الأيمن لـ  $k_4$

$d_5$  الابن الأيمن لـ  $k_5$

هذه البنية توافق شجرة البحث الثنائية المثلى المبينة بالشكل 9.15(ب).

### 2-5.15

حدّد تكلفة بنية شجرة بحث ثنائية مثلى لمجموعة من  $n = 7$  مفاتيح لها الاحتمالات التالية:

$i$	0	1	2	3	4	5	6	7
$p_i$	0.06	0.04	0.06	0.08	0.02	0.10	0.12	0.14
$q_i$	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

### 3-5.15

افترض أنه عوضاً عن الاحتفاظ بالجدول  $wp[i, j]$ ، فإننا حسبنا القيمة  $wp(i, j)$  مباشرة من المعادلة (12.15) في السطر 9 من OPTIMAL-BST واستخدمنا هذه القيمة المحسوبة في السطر 11. كيف يمكن أن يؤثر هذا التغير في زمن التنفيذ للمقارب لـ OPTIMAL-BST؟

### \* 4-5.15

يُعرف كنوث Knuth في المرجع [212] أنه يوجد دائماً جذوراً لشجرات فرعية متلى بحيث  $root[i, j-1] \leq root[i, j] \leq root[i+1, j]$  لكل القيم  $n \geq j > i \geq 1$ . استخدم هذه الحقيقة لتمثيل الإحرائية OPTIMAL-BST كي تُنفذ بزمن  $\Theta(n^2)$ .

## مسائل

### 1-15 أطول مسار بسيط في بيان موجه غير دوار

افترض أن لدينا بياناً موجهاً غير دوار  $G = (V, E)$  مع قيم حقيقية لأوزان الوصلات، وعقدتين متباينتين  $s$  و  $t$ . وصّف نمطاً بالوجهة الديناميكية لإيجاد أطول مسار بسيط من  $s$  إلى  $t$ . ما شكل بيان المسألة الجزئية؟ وما هي فعالية خوارزميةك؟

### 2-15 أطول متتالية جناس عكسي جزئية

الجناس العكسي *palindrome* متتالية عارف غير عالية، من أبجدية معينة، تُقرأ طرّاً وعكساً دون تغيير. من الأمثلة على ذلك في اللغة الإنكليزية، كل المتتاليات التي طولها 1: *civic* و *racecar* و *aibohphobia* (أي رهاب الجناس العكسي).

أنط خوارزمية فعالة لإيجاد أطول متتالية عارف من هذا النوع، تكون متتالية جزئية من متتالية عارف دخل معطاة. على سبيل المثال، إذا كان الدخل *character*، يجب أن تعيد خوارزميةك *carac*. ما هو زمن تنفيذ هذه الخوارزمية؟

### 3-15 مسألة البائع الجوال الإقليدية البيوتونية

في مسألة البائع الجوال الإقليدية *euclidean traveling-salesman problem*، لدينا مجموعة من  $n$  نقطة في المستوي، ونودّ تحديد أصغر جولة مغلقة تربط جميع نقاط  $n$ . يبيّن الشكل 11.15 (أ) الحل لمسألة ب 7 نقاط. المسألة العامة هي NP-صعبة NP-hard، ولذلك فإننا نعتقد بأن حلها يحتاج إلى زمن أكبر من كثير حدودي (انظر الفصل 34).



**الشكل 11.15** سبع نقاط في المسوي، مبيّنة على شبكة واحدة. (أ) أقصر جولة مغلقة، بطول تقريبي 24.89. هذه الجولة ليست بيتونية. (ب) أقصر جولة بيتونية لمجموعة النقاط نفسها. طولها تقريباً 25.58.

يرى J.L. Bentley أن نبسط المسألة بقصر اهتمامنا على **الجولات البيتونية** *bitonic tours*، أي الجولات التي تبدأ من النقطة في أقصى اليسار وتذهب حصراً من اليسار إلى اليمين إلى النقطة في أقصى اليمين، ثم تعود حصراً من اليمين إلى اليسار إلى نقطة البدء. يبيّن الشكل 11-15 (ب) أقصر جولة بيتونية للنقاط الـ 7 نفسها. في هذه الحالة، يمكن إيجاد خوارزمية بزمن كثير حدودي.

وصفّ خوارزمية بزمن  $O(n^2)$  لتحديد جولة بيتونية مثلى. يمكن أن نفترض أنه لا يوجد أي نقطتين لهما إحداثيات  $x$  (الفاصلة) نفسها. (لمنع: آخر المسح من اليسار إلى اليمين، محتفظاً بإمكانات مثلى للجوازين في الجولة.)

#### 15-4 الطباعة المتقنة

لنأخذ بالاعتبار مسألة طباعة فقرّة بإتقان ينط متساوي الفراغات (جميع الحروف العرض نفسه) بطابعة. النص المُدخل هو متالية من  $n$  كلمة بأطوال  $l_1, l_2, \dots, l_n$  مقيّسة بالحرف. نود طباعة هذه الفقرّة بإتقان على عدد من السطور يتسع كل منها لـ  $M$  حرفاً على الأكثر. معيارنا "للإتقان" هو كما يلي: إذا تضمن سطر ما الكلمات من  $l$  إلى  $r$ ، بحيث  $l \leq r$  وتركنا فراغاً واحداً بالضبط بين الكلمات، فإن عدد حروف الفراغ الإضافية في آخر السطر هو  $l - r + 1 - \sum_{k=l}^r l_k$ ، الذي يجب أن يكون غير سالب بحيث يتسع السطر للكلمات. نود تصغير مجموع مكعبات عدد الفراغات الإضافية في أواخر السطور، على كل الأسطر ما عدا السطر الأخير. أعطِ خوارزمية برمجة ديناميكية تُطبع فقرّة من  $n$  كلمة بإتقان على طباعة. حلّ متطلبات زمن التنفيذ وفضاء الذاكرة لخوارزمتك.

#### 15-5 مسافة التحرير

لكي نحول سلسلة محارف محدّدية من نص  $x[1..m]$  إلى سلسلة محارف وجهة  $y[1..n]$ ، يمكننا تطبيق عدة عمليات تحويل. وغرضنا، إذا كانت لدينا  $x$  و  $y$ ، أن نتج سلسلة التحويلات التي تحوّل  $x$  إلى  $y$ .

نستخدم صيغة  $z$ ، نفترضها كبيرة كفاية لتتسع كل الحروف اللازمة، لنضع فيها النتائج للمتوسطة. بداية تكون  $z$  فارغة، وفي النهاية، يجب أن يكون لدينا  $z[j] = y[j]$  لقيم  $j = 1, 2, \dots, n$ . نحتفظ بالأدلة الحالية  $i$  على  $x$  والأدلة  $j$  على  $z$ ، ويُسمح للعمليات أن تغير  $z$  وهذه الأدلة. بداية،  $i = j = 1$ ، والمطلوب فحص كل حرف في  $x$  خلال عملية التحويل، وهذا يعني أنه في نهاية متتالية عمليات التحويل، يجب أن يكون لدينا  $i = m + 1$ .

يمكننا الاختيار من بين ست عمليات تحويل:

نسخ حرف من  $x$  إلى  $z$  بوضع  $z[j] = x[i]$ ، ثم إضافة واحد إلى كل من  $i$  و  $j$ . هذه العملية تفحص  $x[i]$ . استعاضة عن حرف من  $x$  بحرف آخر  $c$  بوضع  $z[j] = c$ ، ثم إضافة واحد إلى كل من  $i$  و  $j$ . هذه العملية تفحص  $x[i]$ .

حذف حرف من  $z$  بزيادة واحد على  $i$ ، وإبقاء  $j$  على حالها. هذه العملية تفحص  $x[i]$ . إدراج حرف  $c$  في  $z$  بوضع  $z[j] = c$ ، ثم زيادة واحد على  $j$ ، وإبقاء  $i$  على حالها. هذه العملية لا تفحص أيًا من محارف  $x$ .

قتل (أي مبادلة) الحرفين التاليين بنسخهما من  $x$  إلى  $z$ ، ولكن بالترتيب المعاكس؛ نفعل ذلك كما يلي:  $z[j] = x[i]$  و  $z[i] = x[j]$ ، ثم بوضع  $i = i + 2$  و  $j = j + 2$ . هذه العملية تفحص  $x[i]$  و  $x[i + 1]$ .

قتل ما تبقى من  $x$  بوضع  $i = m + 1$ . نفحص هذه العملية كل محارف  $x$  التي لم تُفحص بعد. إذا نُفذت هذه العملية كانت بالضرورة العملية النهائية.

كمثال على ذلك، فإن من بين طرق تحويل سلسلة المحارف المصدرية algorithm إلى سلسلة المحارف الوجهة altruistic هي استخدام المتتالية التالية من العمليات، حيث المحارف التي تحتها خط هي  $x[i]$  و  $z[j]$  بعد العملية:

العملية	$x$	$z$
سلسلة المحارف الأولية	algorithm	-
نسخ	al <u>g</u> orithm	a_
نسخ	al <u>g</u> orithm	al_
استعاضة عن الحرف $t$ بـ $c$	al <u>g</u> orithm	alt_
حذف	al <u>g</u> orithm	alt_
نسخ	al <u>g</u> orithm	altr_

altru_	algorithm	إدراج u
altrui_	algorithm	إدراج i
altruiss_	algorithm	إدراج s
altruisti_	algorithm	قتل (مبادلة)
altruistic_	algorithm	إدراج c
altruistic_	algorithm_	قتل

لاحظ أنه توجد عدة متتاليات أخرى لعمليات تحويل `algorithm` إلى `altruimtic`.

إن لكل عملية تحويل تكلفة مرفقة بها. وتعتمد تكلفة عملية ما على خصوصية التطبيق، إلا أننا نفترض أن تكلفة كل عملية هي مقدار ثابت معلوم لنا. نفترض أيضاً أن التكاليف الفردية لعمليات النسخ والاستعاضة أقل من مجموع تكلفة عمليتي الحذف والإدراج؛ وإلا فإنا لا نستخدم عمليات النسخ والاستعاضة. إن تكلفة متالية ما من عمليات التحويل تساوي مجموع تكاليف العمليات الفردية في المتالية. وفي حالة المتتالية السابقة تكون تكلفة تحويل `algorithm` إلى `altruistic` هي:

$$(3 \times \text{تكلفة (نسخ)}) + \text{تكلفة (استعاضة)} + \text{تكلفة (حذف)} + (4 \times \text{تكلفة (إدراج)}) \\ + \text{تكلفة (مبادلة)} + \text{تكلفة (قتل)}.$$

أ. إذا كانت لدينا متاليتان  $x[1..m]$  و  $y[1..n]$  ومجموعة من عمليات التحويل مع تكاليفها، فإن **مسافة التحرير** *edit distance* من  $x$  إلى  $y$  هي تكلفة متالية العمليات الأرخص ثناً التي تحول  $x$  إلى  $y$ . وَصِفَ خوارزمية برمجة ديناميكية توجد مسافة التحرير من  $x[1..m]$  إلى  $y[1..n]$  وتطبع متالية عمليات مثلى. حلّل متطلبات زمن التنفيذ وفضاء الذاكرة لخوارزمتك.

إن مسألة مسافة التحرير تعميمٌ لمسألة رصف متاليتي DNA (انظر على سبيل المثال، المقطع 2.3 في المرجع [310] Meidanis و Setubal). ثمة عدة طرق لقياس التشابه بين متاليتي DNA برصقهما. تتألف إحدى طرق صفّ متاليتين  $x$  و  $y$  من إدراج فراغات في مواضع اعتباطية (لا على التعيين) في المتاليتين (ومنها مواضع عند إحدى النهايات) بحيث يكون للمتتاليات الناتجة  $x'$  و  $y'$  الطول نفسه، ولا يكون هناك فراغ في الموضع نفسه (أي لا يوجد أي موضع  $i$  بحيث يكون كلٌّ من  $x'[i]$  و  $y'[i]$  فراغاً). ثم نعيّن "علامة تقييم" لكل موضع. يستقبل الموضع  $i$  العلامة كما يلي:

- +1 إذا كان  $x'[i] = y'[i]$  وليس فيهما فراغ.
- -1 إذا كان  $x'[i] \neq y'[i]$  وليس فيهما فراغ.
- -2 إذا كان  $x'[i]$  أو  $y'[i]$  فراغاً.

علامة عملية الصرف هي مجموع علامات العمليات للفرقة. على سبيل المثال، إذا كان لدينا المتاليتان  $x = \text{GATCGGCAT}$  و  $y = \text{CAATGTGAATC}$  فإن إحدى عمليات الصرف هي:

G ATCG GCAT  
CAAT GTGAATC  
-+\*+\*+\*+\*+\*+

يشير + تحت موضع ما إلى علامة تساوي +1 لذلك للموضع، ويشير - إلى علامة -1 و \* إلى علامة -2، بحيث يكون لعملية الصرف هذه علامة إجمالية  $-4 = -4 \cdot 2 - 2 \cdot 1 - 6 \cdot 1$ .

ب. اشرح طريقة صوغ مسألة إيجاد عملية صرفٍ مثلى باعتبارها مسألة مسافة تحرير باستخدام مجموعة جزئية من عمليات التحويل: نسخ واستعاضة وحذف وإدراج ومبادلة وقتل.

### 6-15 التخطيط لحفلة لشركة

يعمل الأستاذ ستewart مستشارًا لمدير شركة تخطط لحفلة في الشركة. وللشركة بنية هرمية (تراتبية) أي إن علاقات المشرفين تؤلف شجرة جذرها الرئيس. وقد أسند مكتب الموظفين إلى كل موظف ترتيبًا بحسب درجة مرحه، يمثل عددًا حقيقيًا. ولجعل الحفلة ممتعة لكل الحضور، لا يريد الرئيس أن يحضر الموظف مع مسؤوله المباشر.

أعطي الأستاذ ستewart الشجرة التي تُوصف بنية الشركة باستخدام التمثيل: الابن الأيسر والأخ الأيمن المُوصف في المقطع 4.10. وتُعمل كل عقدة من الشجرة، إضافة إلى المؤشرات، اسم للموظف وترتيبه في المرح. وصُف خوارزمية لتأليف قائمة الضيوف، بحيث يكون مجموع تقديرات الضيوف المرحمة أعظميًا. حلّل زمن تنفيذ خوارزمتك.

### 7-15 خوارزمية ليعربي

يمكننا استخدام الوبجة الديناميكية على بيانٍ موجه  $G = (V, E)$  لتعرّف الكلام. نوضح على كل وصلة  $(u, v) \in E$  لصيقة بصوت  $\sigma(u, v)$  من مجموعة متتهية  $\Sigma$  من الأصوات. إن البيان مع لصيقاته هو نموذج صوري لشخص يتكلم لغة محددة. يبدأ كل مسارٍ في البيان من عقدة مميزة  $v_0 \in V$  توافق متتاليةً ممكنةً من الأصوات التي ينتجها النموذج. تُعرّف لصيقة مسارٍ موجه بأنها تتابع للصقات الوصلات على المسار.

أ. وصّف خوارزمية فعالة تستطيع - بوجود بيانٍ  $G$  على وصلاته لصقات، وعقدة مميزة  $v_0$  ومتتالية من الحارف  $s = (\sigma_1, \sigma_2, \dots, \sigma_k)$  من  $\Sigma$  - أن تعيد مسارًا في  $G$  يبدأ من  $v_0$  ولصيقته  $s$  في حال وجود مثل هذا المسار. وإلا فيجب أن تعيد الخوارزمية عبارة NO-SUCH-PATH. حلّل زمن تنفيذ خوارزمتك. (تلميح: يمكن أن تجد بعض المفاهيم المفيدة في الفصل 22.)

الآن افترض أننا أعطينا كل وصلة  $(u, v) \in E$  احتمالاً غير سالب  $p(u, v)$  لاحتياز الوصلة  $(u, v)$  من



العقدة  $u$ ، متحة بذلك الصوت للوافق. إن مجموع احتمالات الوصلات التي تنطلق من أي عقدة يساوي 1. ويعرف احتمال المسار بأنه جداء احتمالات وصلاته. وبمكتنا النظر إلى احتمال مسار يبدأ من  $v_0$  على أنه احتمال "سُتَر عشوائي" يبدأ من  $v_0$  ويتبع للمسار المحدد، حيث نختار عشوائيًا الوصلة المطلوبة، ونُدع العقدة  $u$  وفقًا لاحتمالات الوصلات المتاحة التي تغادر  $u$ .

ب. وسع إحداثك على الجزء (أ) بحيث إذا أعيد مسار ما فهو للمسار الأكثر احتمالاً الذي يبدأ من  $v_0$  ولصيقته  $s$ . حلّ زمن تنفيذ خوارزمتك.

### 8-15 ضغط الصورة بنقش عروق

لتكن لدينا صورة ملونة تتكون من صحيفة  $A[1..m, 1..n]$  أبعادها  $m \times n$  من البكسلات pixels (أو العناصر)، حيث يُعدّد كل بكسل ثلاثية من كثافات الألوان: الأحمر والأخضر والأزرق (RGB). افترض أننا نودّ ضغط هذه الصورة ضغطاً بسيطاً. وبالتحديد، نريد أن نخذف بكسلًا من كل سطر من الأسطر  $m$  بحيث تصبح الصورة بكاملها أضيق ببكسل واحد. ولكن، لتحتب آثار التشوه البصري، نطلب أن يقع البكسلان، اللذان نخذفهما من سطرين متجاورين، في العمود نفسه أو في عمودين متجاورين؛ تشكل البكسلات المحذوفة "جرفًا أو درزة" من أعلى سطر إلى أدنى سطر حيث تكون البكسلات المتتالية في العرق متجاورة شاقوليًا أو قطريًا.

أ. بين أن عدد مثل هذه العروق للمكنة يزداد أسّيًا في  $m$  على الأقل بافتراض أن  $n > 1$ .

ب. افترض الآن أننا حسبنا مع كل بكسل  $A[i, j]$  قياس تمزق  $d[i, j]$  قيمته حقيقية، مشيرة إلى مدى التمزق الحاصل من حذف البكسل  $A[i, j]$ . وبلاخط بالبدئية أنه كلما كان قياس تمزق البكسلات أصغر، كان البكسل أكثر شبهاً بمجاوراته. افترض أيضًا أننا عرفنا قياس تمزق العرق على أنه مجموع تمزقات بكسلاته.

أعط خوارزمية لإيجاد العرق ذي قياس التمزق الأصغر. ما مدى فعالية خوارزمتك؟

### 9-15 قطع متتاليات محرفية

تتيح بعض لغات معالجة المتتاليات المحرفية للمبرمج قُطْع متتالية محرفية إلى قطعتين. ولأن هذه العملية تُنسخ المتتالية المحرفية، فإنها تكلف  $n$  وحدة زمن لقطع متتالية من  $n$  حرفًا إلى قطعتين. افترض أن المبرمج يريد قطع المتتالية إلى عدة قطع؛ إن الترتيب الذي يحصل وفقه القطع يمكن أن يؤثر في الزمن الكلي المستغرق. على سبيل المثال، افترض أن المبرمج يريد أن يُقطع متتالية من 20 حرفًا بعد الحارفين 2 و 8 و 10 (بترقيم الحارفين بالترتيب للتصاعد من اليسار، بدءًا من الرقم 1). فإذا بَرَزِمَج القُطْع بحيث يحدث بالترتيب من اليسار إلى اليمين، فإن القطع الأول يكلف 20 وحدة زمن، والثاني يكلف 18 وحدة زمن (قطع المتتالية من الحرف 3 إلى

الحرف 20 عند الحرف 8). ويكلف القطع الثالث 12 وحدة زمن، ويكون المجموع 50 وحدة زمن. أما إذا بَرَزَجَ القَطْعُ بحيث يحدث بالترتيب من اليمين إلى اليسار، فتعدها سيكلف القطع الأول 20 وحدة زمن، والثاني 10 وحدات زمن، والثالث 8 وحدات زمن، ويكون المجموع 38 وحدة زمن. وفي ترتيب آخر مختلف، يمكن أن يقطع للمرجع عند الحرف 11 أولاً (بتكلفة 20)، ثم يقطع القطعة اليسرى عند الحرف 2 (بتكلفة 8)، وأخيراً القطعة اليمنى عند الحرف 10 (بتكلفة 12)، ويكون إجمالي التكلفة 40.

صمّم خوارزمية، إذا أُعطيت عددًا من الحواف يجري بعدها القَطْع، تُحدّد الطريقة ذات التكلفة الدنيا لترتيب عمليات القطع هذه. وعلى نحو صوري أكثر، إذا أُعطيت متتالية عرقية  $S$ ، مؤلفة من  $n$  حروفًا، وصيغة  $L[1..m]$  تتضمن نقاط القطع، احسب أدنى تكلفة لمتتالية القطع، مع متتالية القطع التي تُنتج بهذه التكلفة.

#### 10-15 تخطيط استراتيجي استثمار

تساعدك معرفتك بالخوارزميات على الحصول على عمل مثير في شركة Acme Computer Company مع مكافأة 10,000 دولار عند التوقيع. قرّرت استثمار هذه الأموال بهدف جعل دخلك أعظميًا في نهاية 10 سنوات. وقرّرت استخدام شركة Amalgamated Investment Company لإدارة استثماراتك. تُطلب هذه الشركة احترام القواعد التالية: إذا تقدم  $n$  استثمارًا مختلفًا، مرقمة من 1 إلى  $n$ . وفي كل عام  $t$ ، يزودك الاستثمار  $i$  بمعدل دخل (ربح)  $r_{it}$ . وتعتبر آخر: إذا استثمرت  $d$  دولارًا بالاستثمار رقم  $i$  من العام  $t$ ، فإنك تحصل على  $d r_{it}$  في نهاية العام  $t$ . ومعدلات الدخل مضمونة، أي إنك تحصل على كل معدلات الدخل على مدى السنين العشر التالية لكل استثمار. ويمكنك اتخاذ قرارات استثمار مرة واحدة في السنة. فإذا قرّرت أن تترك أموالك في المجموعة نفسها من الاستثمار عاتقَين متالين، عليك أن تسدد رسمًا مقداره  $f_1$  دولارًا، أما إذا قررت تحويل أموالك إلى مجموعة استثمار مختلفة فإنك تسدد رسمًا مقداره  $f_2$  دولارًا، حيث  $f_1 > f_2$ .

أ. تسمح لك المسألة، حسبما ذُكرت، باستثمار أموالك في عدة استثمارات كل سنة. أثبت أنه توجد استراتيجية مثلى، تكمن في وضع جميع الأموال في استثمار واحد في كل سنة. (تذكّر أن استراتيجية الاستثمار المثلى تجعل المبلغ بعد 10 سنوات أعظميًا، وهي غير معيّنة بأيّ أغراضٍ أخرى، كتقليص الأخطار إلى الحدود الدنيا).

ب. برهن أن مسألة تخطيط استراتيجية الاستثمار المثلى تكشف عن بنية جزئية مثلى.

ت. صمّم خوارزمية تخطيط استراتيجية استثمار المثلى. ما زمن تنفيذ خوارزمتك؟

ث. افترض أن شركة Amalgamated Investments فرضت قيودًا إضافية بحيث لا يمكنك، في كل لحظة،

وضع أكثر من 15,000 دولار في استثمار واحد. يَبَيَّنُ أن مسألة تعظيم دخلك بعد 10 سنوات لم تعد تُظهر بنية جزئية مثلى.

### 11-15 تخطيط المخزون

تُنتج شركة Rinky Dink Company آلات لتحديد سطوح للزائج الجليدية. يتغير الطلب على مثل هذه المنتجات من شهر لآخر، ولذلك تحتاج الشركة إلى تطوير استراتيجية لتخطيط تصنيعها في ضوء الطلب المتقلب والمتوقع في الوقت نفسه. ترغب الشركة في تصميم خطة للأشهر الـ  $n$  التالية. ولكل شهر  $i$  تعلم الشركة عدد الطلبات  $d_i$  أي عدد الآلات التي ستبيعها. ليكن  $D = \sum_{i=1}^n d_i$  مجموع الطلبات على مدى الأشهر  $n$  التالية. تحتفظ الشركة بموظفيها الذين يعملون بدوام كامل للقيام بالعمل اللازم لتصنيع  $m$  آلة في الشهر. إذا احتاجت الشركة إلى تصنيع أكثر من  $m$  آلة في شهر معين، يمكنها استئجار عمال بدوام جزئي، بتكلفة تبلغ  $c$  دولارًا للآلة الواحدة. وإضافة إلى ذلك، إذا بقي لدى الشركة، في نهاية الشهر، أي آلات لم تُبَع، فإنها تسدد تكاليف تخزينها. تُعطي تكلفة الاحتفاظ بـ  $j$  آلة كدالة  $h(j)$  للقيم  $j = 1, 2, \dots, D$  حيث  $h(j) \geq 0$  في حالة  $j \leq D$  ولدينا أيضًا  $h(j) \geq h(j+1)$  في حالة  $1 \leq j \leq D-1$ .

أعطِ خوارزمية لحساب خطة للشركة تُخفِّض التكاليف إلى حدودها الدنيا في الوقت الذي تحقق فيه جميع الطلبات. يجب أن يكون زمن التنفيذ أكثر حدود في  $n$  و  $D$ .

### 12-15 التعاقد مع لاعبي البيسبول الأحرار

افترض أنك مدير عام لفريق بيسبول من الفئة الأولى. ربما تحتاج، خارج الموسم، إلى التعاقد مع لاعبين أحرار لفريقك. ولقد أعطاك مالك الفريق ميزانية  $X$  دولارًا للإنفاق على اللاعبين الأحرار. وسمح لك أن تنفق أقل من  $X$  دولارًا بالمجموع، ولكنه سيستغي عن خدماتك إذا أنت تجاوزت في الإنفاق  $X$  دولارًا. لديك  $N$  موقفًا مختلفًا، ولكل موقع يوجد  $P$  لاعبًا حُرًا متاحًا للعب ذلك للموقع.<sup>8</sup> ولما كنت لا تريد أن تنقل قائمتك بعدد كبير من اللاعبين لموقع ما، فبإمكانك أن تتعاقد مع لاعب واحد على الأكثر لكل موقع. (إذا لم تتعاقد مع أي لاعب لموقع معين، فليكن أن يُبقي على لاعبيك لذلك الموقع.)

ولكي نحدد قيمة لاعب ما، فإنك تقرر استخدام إحصائيات  $saber$ <sup>9</sup> تُعرف بـ "VORP" أو "value over replacement player" أي "القيمة عند تبديل لاعب". فلاعب ذو قيمة VORP عالية أغلى ثمنًا من

<sup>8</sup> مع أنه يوجد تسعة مواقع لفريق البيسبول، فإن  $N$  لا تساوي بالضرورة 9 لأن بعض المديرين طرقًا خاصة للتفكير في المواقع. فمثلاً، قد يعتبر المدير أن المزمة اليمنى والمزمة اليسارية مواقع منفصلة، وكذلك مزمة الاستهلال ومزمة الاحتياط الذين يمكنهم المزمة في عدة جولات، ومزمة الاحتياط الذين يرمون عادة في جولة واحدة على الأكثر.

<sup>9</sup> هو تطبيق التحليل الإحصائي على سجلات البيسبول. وهو يتيح عدة طرق لمقارنة القيم النسبية لأفراد اللاعبين.

لاعب ذي VORP منخفضة. إلا أن اللاعب ذا VORP عالية ليس بالضرورة أغلى ثمنًا للتعاقد من لاعب ذي VORP منخفضة، إذ إن ثمة عوامل أخرى غير ثمن اللاعب تحدد تكلفة التعاقد معه. نأخذ في الاعتبار ثلاثة أمور بشأن كل لاعب موجود:

- موقع اللاعب
- تكلفة تعاقد اللاعب
- قيمة VORP للاعب.

صنم خوارزمية تجعل مجموع VORP للاعبين الذين تتعاقد معهم أعظميًا، على ألا يتجاوز مجموع إنفاقك  $X$  دولارًا. يمكنك أن تفترض أن تعاقد كل لاعب من مضاعفات 100,000 دولار. ويجب أن يكون خرج خوارزمتك: مجموع قيمة VORP للاعبين الذين تتعاقد معهم، ومجموع المال الذي تنفقه، وقائمة باللاعبين الذين وقع اختيارك عليهم للتعاقد. حلّ متطلبات زمن التنفيذ والذاكرة المطلوبة لخوارزمتك.

## ملاحظات الفصل

استهلّ R. Bellman الدراسة المنهجية للبرمجة الديناميكية في عام 1955. وتشير كلمة "برمجة" - هنا وفي البرمجة الخطية معًا - إلى استخدام طريقة الحل المُحدّول. ومع أن تقنيات الأمثلة التي تتضمن عناصر البرمجة الديناميكية كانت معروفة من قبل، فقد زوّد يلمان هذا المجال بأساسي رياضي متين [37].

وصنف Park و Galil [125] خوارزميات البرمجة الديناميكية بحسب حجم الجدول وعدد عناصر الجدول الأخرى التي يعتمد عليها كل عنصر. وهما يسميان خوارزمية البرمجة الديناميكية  $td/eD$  إذا كان حجم جدولها  $O(n^2)$  وكان كل عنصر يعتمد على  $O(n^2)$  عنصرًا آخر. فمثلًا، خوارزمية جداء المصفوفات الواردة في المقطع 2.15 هي  $2D/1D$ ، وخوارزمية أطول متتالية عرقية مشتركة الواردة في المقطع 4.15 هي  $2D/OD$ . وأعطى Hu و Shing [182, 183] خوارزمية بزمن  $O(n \lg n)$  لمسألة جداء سلسلة مصفوفات.

ويبدو أن الخوارزمية التي تنفذ بزمن  $O(mn)$  لمسألة إيجاد أطول متتالية جزئية مشتركة هي خوارزمية شعبة. فقد طرح Knuth [71] تساؤلًا عن إمكان وجود خوارزميات بزمن أقل من الرتبة التربيعية (من الدرجة الثانية) subquadratic algorithms لمسائل LCS. وأجاب Masek و Paterson [244] عن هذا السؤال بالإيجاب، وذلك بإعطاء خوارزمية تُنفذ بزمن  $O(mn/\lg n)$ ، حيث  $n \leq m$  وللتساليات مأخوذة من مجموعة محدودة الحجم. وفي الحالة الخاصة، التي لا يظهر فيها أي عنصر أكثر من مرة واحدة في متتالية الدخول، بيّن Szymanski [326] كيف يمكن حل للمسألة بزمن  $O((n+m) \lg(n+m))$ . وينطبق كثيرٌ من هذه النتائج على مسألة حساب مسافات تحرير سلسلة محارف (المسألة 5-15).

ثمة مقالة قديمة كتبها Gilbert و Moore [133] عن الترميزات الانشائية المتغيرة الطول، كان لها تطبيقات في بناء شجرات بحث ثنائية مثلثي للحالة التي تكون فيها قيم جميع الاحتمالات  $p_i$  تساوي 0؛ تضمنت هذه المقالة خوارزمية بزمن  $O(n^3)$ . وكذلك قدّم Aho و Hopcroft و Ullman [5] الخوارزمية الواردة في المقطع 5.15. ويعود التمرين 4-5.15 إلى Knuth [212]. على حين ابتكر Tucker و Hu [184] خوارزمية للحالة التي تكون فيها الاحتمالات  $p_i$  مساوية للصفر تُستخدم زمنًا  $O(n^2)$  وفضاءً  $O(n)$  وفيما بعد قلّص كوث [211] الزمن إلى  $O(n \lg n)$ .

وتعود المسألة 8-15 إلى Avidan و Shamir [27]، اللذين وضعوا فيديو رائعًا على الويب يبيّن هذه التقنية في ضغط الصورة.

تُمرّ خوارزميات مسائل الأمثلة عادةً بمتتالية من الخطوات، مع مجموعة من الخيارات عند كل خطوة. وفي كثير من مسائل الأمثلة، يُعدّ استبعاد البرمجة الديناميكية لتحديد أفضل الخيارات إفراطاً؛ على أن ثمة خوارزميات أبسط وأكثر فاعلية يمكن أن تفنّي بالفرض. تُختار **الخوارزمية الشرهة** *greedy algorithm* دائماً الخيار الذي يبدو أنه الأفضل في تلك اللحظة. أي إنها تأخذ الخيار الأفضل محلياً، على أمل أن يقود هذا الخيار إلى حلٍّ أمثلٍ شامل. يستكشف هذا الفصل مسائل الأمثلة التي يمكن حلّها بخوارزميات شرهة. وقبل قراءة هذا الفصل، ينبغي قراءة ما ورد عن البرمجة الديناميكية في الفصل 15، وبوجهٍ خاص المقطع 3.15.

إن الخوارزميات الشرهة لا تتوصل دائماً إلى حلولٍ مثلى، لكنها تتوصل إليها في الكثير من المسائل. بدايةً، سندرس في المقطع 1.16، مسألة بسيطة إلا أنها غير تافهة؛ وهي مسألة اختيار النشاطات. وهي مسألة تُحسب لها خوارزمية شرهة حلّاً أمثلٍ بفعالية. سنصل إلى الخوارزمية الشرهة باعتماد طريقة البرمجة الديناميكية أولاً، ثم نبيّن أن بإمكاننا دائماً القيام بخيارات شرهة للوصول إلى حلٍّ أمثل. يستعرض المقطع 2.16 العناصر الأساسية للنهج الشره، معطياً نمحاً أبسط لبرهان صحة الخوارزميات الشرهة. ويقدم المقطع 3.16 تطبيقاً هاماً للنقليات الشرهة: تصميم أرمزة هوفمان Huffman لضغط الملفات. وننتخّص في المقطع 4.16 جزءاً نظرياً، يؤسس للنبي التوافقية المسماة "كيانات مصفوفية *metroid*"، التي تُنتج الخوارزمية الشرهة دائماً حلّاً أمثل لها. أخيراً، يبيّن المقطع 5.16 تطبيق الكيانات المصفوفية باستخدام مسألة جدولة مهام في واحدة الزمن مع مدد انتهاء وعقوبات.

إن الطريقة الشرهة قوية وتعمل جيّداً على مجال واسع من المسائل. ستقدم الفصول اللاحقة عدة خوارزميات يمكن رؤيتها على أنها تطبيقات للطريقة الشرهة، ومنها خوارزميات شجرة المسح الصغرى (الفصل 23)، وخوارزمية Dijkstra لأقصر الطرق من مصدر وحيد (الفصل 24)، وكسبية Chvátal الشرهة لتغطية مجموعة (الفصل 35). ومع أنه يمكن قراءة هذا الفصل والفصل 23 على نحو مستقل، ولكن قد تجد أن قراءة أحدهما معاً مفيدة.

## 1.16 مسألة اختيار النشاطات

مثالنا الأول هو مسألة جدولة عدة نشاطات متنافسة تتطلب استخدام مورد مشترك استخدامًا حصريًا، والهدف هو اختيار مجموعة نشاطات متوافقة فيما بينها وذات حجم أعظم. افترض أن لدينا مجموعة من  $n$  نشاطات  $activities = \{a_1, a_2, \dots, a_n\}$   $S$  تريد جميعها استخدام مورد ما (قاعة محاضرات مثلاً)، يمكن أن يُخدم نشاطاً واحداً في كل مرة. لكل نشاط  $a_i$  لحظة بداية  $s_i$  ولحظة انتهاء  $f_i$ ،  $finish\ time$  حيث  $0 \leq s_i < f_i < \infty$ . إذا جرى اختيار النشاط  $a_i$  فإنه يحدث خلال المجال الزمني نصف المفتوح  $(s_i, f_i)$ . نقول عن نشاطين  $a_i$  و  $a_j$  أنهما متوافقان *compatibles* إذا لم يكن المجال  $(s_i, f_i)$  و  $(s_j, f_j)$  متراكبين (overlap؛ أي إن النشاطين  $a_i$  و  $a_j$  متوافقان إذا كان  $s_i \geq f_j$  أو  $s_j \geq f_i$ ). إن مسألة اختيار المتوافقة فيما بينها، لندرس، على سبيل المثال، المجموعة التالية  $S$  من النشاطات، التي جرى فرزها بحسب الترتيب المتزايد باطراد للحظات الانتهاء.

$$f_1 \leq f_2 \leq f_3 \leq \dots \leq f_{n-1} \leq f_n. \quad (1.16)$$

(سنرى لاحقاً الفائدة التي يحققها هذا الافتراض.) لندرس مثلاً مجموعة النشاطات التالية  $S$ :

$i$	1	2	3	4	5	6	7	8	9	10	11
$s_i$	1	3	0	5	3	5	6	8	8	2	12
$f_i$	4	5	■	7	9	9	10	11	12	14	16

في هذا المثال، تتألف المجموعة  $\{a_3, a_9, a_{11}\}$  من نشاطات متوافقة فيما بينها، ولكنها ليست مجموعة جزئية غُطّيت، لأن المجموعة  $\{a_1, a_4, a_8, a_{11}\}$  أكبر منها. والواقع أن المجموعة  $\{a_1, a_4, a_8, a_{11}\}$  هي مجموعة جزئية غُطّيت من النشاطات المتوافقة؛ ولغة مجموعة غُطّيت أخرى هي  $\{a_2, a_4, a_9, a_{11}\}$ . سنحل هذه المسألة بعدة خطوات؛ نبدأ بالتفكير في حلّ هذه المسألة بالبرمجة الديناميكية. في هذا الحل، نأخذ بالحسبان عدة خيارات حين نحدّد المسائل الجزئية التي يجب استخدامها في الحل الأمثل. سنلاحظ لاحقاً أن علينا أن نأخذ بالحسبان خياراً واحداً - وهو الخيار الشر - وأتينا حين نعتّمه، فإن مسألة جزئية واحدة تبقى. اعتماداً على هذه الملاحظات فإننا سنطوّر خوارزمية غُذوية شرية لحلّ مسألة جدولة النشاطات. وستستمرّ إجرائية تطوير الحل الشرية بتحويل الخوارزمية الغُذوية إلى خوارزمية تكرارية. ومع أن الخطوات التي سنسرر وقّعتها في هذه القطع هي أكثر تفصيلاً مما هو معتاد عند تطوير خوارزمية شرية، إلا إنها تبين العلاقة بين الخوارزميات الشرية والبرمجة الديناميكية.

### البنية الجزئية المثلى في مسألة اختيار النشاطات

يمكننا التحقق بسهولة من أن مسألة اختيار النشاطات تُظهر بنيةً جزئيةً مثلى. لنرمز بـ  $S_{ij}$  إلى مجموعة النشاطات التي تبدأ قبل أن ينتهي النشاط  $a_i$ ، وتنتهي قبل بدء النشاط  $a_j$ . ولنفترض أننا نود إيجاد مجموعة غطى من النشاطات المتوافقة فيما بينها في  $S_{ij}$ ، ولنفترض أيضاً أن هذه المجموعة الغطى هي  $A_{ij}$ ، وأنها تتضمن نشاطاً ما  $a_k$ . فإذا ضُمت  $a_k$  في حلٍّ أمثل، فإننا أمام مسألتين جزئيتين: إيجاد النشاطات المتوافقة فيما بينها في  $S_{ik}$  (النشاطات التي تبدأ بعد انتهاء النشاط  $a_i$  وتنتهي قبل بدء النشاط  $a_k$ )، وإيجاد النشاطات المتوافقة فيما بينها في  $S_{kj}$  (النشاطات التي تبدأ بعد انتهاء النشاط  $a_k$  وتنتهي قبل بدء النشاط  $a_j$ ). ولكن  $A_{ik} = A_{ij} \cap S_{ik}$  و  $A_{kj} = A_{ij} \cap S_{kj}$ ، بحيث تتضمن  $A_{ik}$  النشاطات في  $A_{ij}$  التي تنتهي قبل بدء  $a_k$ ، وتتضمن  $A_{kj}$  النشاطات في  $A_{ij}$  التي تبدأ بعد انتهاء  $a_k$ . وبذلك يكون لدينا  $A_{ij} = A_{ik} \cup \{a_k\} \cup A_{kj}$ ، وتتكوّن المجموعة  $A_{ij}$  (ذات الحجم الأعظم من النشاطات المتوافقة فيما بينها في  $S_{ij}$ ) من  $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1$  نشاطاً.

نُظهر حجةً القص واللمص الاعتيادية وجوب أن يتضمن الحل الأمثل  $A_{ij}$  حلولاً مثلى لكلٍّ من المسألتين الجزئيتين  $S_{ik}$  و  $S_{kj}$ . فلو كان بإمكاننا إيجاد مجموعة  $A'_{kj}$  من النشاطات المتوافقة فيما بينها في  $S_{kj}$  حيث  $|A'_{kj}| > |A_{kj}|$ ، لكان بوسعنا استخدام  $A'_{kj}$  عوضاً عن  $A_{kj}$  في حل المسألة الجزئية  $S_{ij}$ . وبذلك نكون قد بنينا مجموعة من النشاطات المتوافقة فيما بينها  $|A_{ij}| = |A_{ik}| + |A_{kj}| + 1 > |A_{ik}| + |A'_{kj}| + 1$ ، وهذا يناقض كون  $A_{ij}$  حلاً أمثلاً. وتطبيق حجة النظر على النشاطات في  $S_{ik}$ .

توحي هذه الطريقة في وصف بنية الحل الأمثل إلى أن بإمكاننا حل مسألة اختيار النشاطات بالبرمجة الديناميكية. فإذا رمزنا إلى حجم الحل الأمثل للمجموعة  $S_{ij}$  بـ  $c[i, j]$ ، فيمكننا كتابة التكرار

$$c[i, j] = c[i, k] + c[k, j] + 1 .$$

بالطبع، إذا لم نكن نعلم أن حلاً أمثلاً للمجموعة  $S_{ij}$  يتضمن النشاط  $a_k$  لوجب علينا فحص جميع النشاطات في  $S_{ij}$  لإيجاد نشاط نختاره، بحيث:

$$c[i, j] = \begin{cases} 0 & \text{if } S_{ij} = \emptyset , \\ \max_{a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & \text{if } S_{ij} \neq \emptyset . \end{cases} \quad (2.16)$$

يمكننا بعدها تطوير خوارزمية فعّالة لاستدكار هذا الحل، أو يمكننا العمل صعوداً ملء عناصر الجدول مع تقدم الحل. ولكن في هذه الحالة نكون قد تجاوزنا خاصية هامة أخرى لمسألة اختيار النشاطات، يمكننا أن نستفيد منها كثيراً.

### القيام بالخيار الشره

ماذا لو كان بإمكاننا اختيار نشاط لإضافته إلى حلنا الأمثل دون أن نكون ملزمين بحل جميع المسائل الجزئية



سلفاً؟ إن ذلك سيحتجنا دراسة جميع الخيارات الموجودة في التكرار (2.16). في الحقيقة، نحتاج في مسألة اختيار النشاطات إلى دراسة خيارٍ وحيدٍ هو: الخيار الشرية.

ماذا نعني بالخيار الشرية في مسألة اختيار النشاطات؟ يقترح الحدس أن نختار نشاطاً يترك المورد متاحاً لأكبر عددٍ ممكن من النشاطات الأخرى. والآن، يجب أن يكون أول النشاطات انتهاء أحد النشاطات التي نختارها. وهنا يقتضي أن نختار من  $S$  نشاطاً الأولي انتهاءً، لأنه سيرك المورد متاحاً لأكبر عددٍ ممكن من النشاطات الأخرى. (إذا وُجد أكثر من نشاط في  $S$  له الانتهاء الأبعد، أمكننا اختيار أيٍّ منها.) وبعبارةٍ أخرى، لما كانت النشاطات مفروزة بحسب ترتيب لحظات انتهائها للزيادة بترتيب، فإن الخيار الشرية هو النشاط  $a_1$ . على أن اختيار النشاط الذي ينتهي أولاً ليس الطريقة الوحيدة للقيام بالخيار الشرية لهذه المسألة. يُطلب إليك في التمرين 1.16-3 استكشاف إمكانيات أخرى.

إذا قمنا بالخيار الشرية، يبقى علينا حلُّ مسألة جزئية واحدة فقط: إيجاد النشاطات التي تبدأ بعد انتهاء  $a_1$ . لماذا لا يجب علينا إيجاد النشاطات التي تنتهي قبل بدء  $a_1$ ؟ لدينا  $s_1 < f_1$  و  $f_1$  هو الأبعد انتهاءً من أي نشاط، لذلك، لا يمكن لأي نشاط أن تكون لحظة انتهائه أقل أو تساوي  $s_1$ . وهكذا، فإن جميع النشاطات المتوافقة مع النشاط  $a_1$  يجب أن تبدأ بعد انتهاء  $a_1$ .

يُضاف إلى ذلك، أننا برهنا سابقاً أن مسألة اختيار النشاطات تُظهر بنيةً جزئيةً مُثلّية. لنكن  $S_K = \{a_i \in S : s_i \geq f_K\}$  مجموعة النشاطات التي تبدأ بعد انتهاء  $a_K$ . إذا قمنا بالاختيار الشرية للنشاط  $a_1$ ، فإن  $S_1$  تبقى المسألة الجزئية الوحيدة التي علينا حلها.<sup>1</sup> نستنتج من البنى الجزئية المُثلّية أنه إذا كان  $a_1$  في الحل الأمثل، فإن الحل الأمثل للمسألة الأصلية يتكوّن من النشاط  $a_1$  ومن كل النشاطات في حل أمثل للمسألة الجزئية  $S_1$ .

يبقى سؤال كبير واحد: هل حدسنا صحيح؟ هل الخيار الشرية - الذي نختار فيه النشاط الذي ينتهي أولاً - هو دائماً جزء من حل أمثل ما؟ تبيّن المبرهنة التالية أنه كذلك.

### مبرهنة 1.16

لنكن  $S_K$  أية مسألة جزئية غير خالية، وليكن  $a_m$  نشاطاً في  $S_K$  له أبكر لحظة انتهاء، عندها يكون  $a_m$  مُضماً في مجموعة جزئية ذات حجم أعظم من نشاطات  $S_K$  المتوافقة فيما بينها.

**البرهان** لنكن  $A_K$  مجموعة جزئية ذات حجم أعظم من النشاطات المتوافقة فيما بينها في  $S_K$ . وليكن  $a_j$  النشاط في  $A_K$  ذا لحظة الانتهاء الأبعد. إذا كان  $a_j = a_m$ ، فقد تمثّل المطلوب، لأننا نبيّن أن  $a_m$  هي في مجموعة جزئية ذات حجم أعظم من النشاطات المتوافقة فيما بينها من  $S_K$ . أما إذا كان  $a_j \neq a_m$ ، فنفترض

<sup>1</sup> نشير أحياناً إلى المجموعات  $S_K$  على أنها مسائل جزئية بدلاً من كونها مجموعات نشاطات. وسيتضح دائماً من السياق إذا كنا نشير بـ  $S_K$  إلى مجموعات النشاطات أو إلى مسائل جزئية مداخلها هذه المجموعات.

أن المجموعة  $A'_k = A_k - \{a_f\} \cup \{a_m\}$  هي  $A_k$  مع الاستعاضة عن  $a_m$  بـ  $a_f$ . فتكون النشاطات في  $A'_k$  منفصلة، وذلك لأن النشاطات في  $A_k$  منفصلة، ويكون  $a_f$  هو أول نشاط ينتهي في  $A_k$ ، و  $f_m \leq f_j$ ، ولما كان  $|A_k| = |A'_k|$ ، فإن  $A'_k$  هي مجموعة جزئية ذات حجم أعظم من النشاطات المتوافقة فيما بينها من  $S$ ، وهي تتضمن  $a_m$ .

وهكذا، نرى أنه على الرغم من أننا قد نكون قادرين على حل مسألة اختيار النشاطات باستخدام البرمجة الديناميكية، إلا أننا لسنا ملزمين بها. (إضافة إلى أننا لم نختبر بعد إذا كان لمسألة اختيار النشاطات مسائل جزئية متراكبة.) عوضاً عن ذلك، يمكننا اختيار النشاط الذي ينتهي أولاً تكرارياً، والإبقاء فقط على النشاطات المتوافقة معه، ونكرر ذلك حتى انتهاء النشاطات. يضاف إلى ذلك، أنه بسبب اختيارنا النشاط ذات لحظة الانتهاء الأبعد دائماً، فيجب أن تكون لحظات انتهاء النشاطات التي نختارها متزايدة تماماً. يمكننا إذن أن ندرس إمكان أخذ كل نشاط مرة واحدة خلال عملنا، وذلك تبعاً للترتيب المتزايد لبطاريات اللحظات انتهاء النشاطات.

لا تحتاج الخوارزمية التي تحل مسألة اختيار النشاطات إلى أن تعمل صعودياً، كما هو الحال في خوارزمية البرمجة الديناميكية المعتمدة على الجداول. عوضاً عن ذلك، يمكننا أن نعمل نزولياً، باختيار نشاط ووضعه في الحل الأمثل، ثم نحل المسألة الجزئية للمثلة باختيار النشاطات من بين تلك المتوافقة مع النشاطات المختارة سابقاً. للخوارزميات الشرة عادةً هذا التصميم النزولي: حدّد الخيار ثم حلّ مسألة جزئية، وذلك عوضاً عن التقنية الصعودية القائمة على حلّ المسائل الجزئية قبل تحديد الخيار.

### خوارزمية غوّذية شرة

الآن بعد أن عرفنا كيف نتجاوز طريقة البرمجة الديناميكية، وبدلاً عن استخدام خوارزمية شرة نزولية، يمكننا كتابة إجراء غوّذي مباشر لحل مسألة اختيار النشاطات. يأخذ الإجراء **RECURSIVE-ACTIVITY-SELECTOR** لحظات بدء النشاطات ولحظات انتهائهما، ممثلة في صفتين  $s$  و  $f$ ،<sup>2</sup> وللمؤشر  $k$  الذي يعرف المسألة الجزئية  $S_k$  الواجب حلّها، و  $n$  حجم للمسألة الأصلية. تعيد هذه الخوارزمية مجموعة ذات حجم أعظم من النشاطات المتوافقة فيما بينها في  $S_k$ . نفترض أن نشاطات الدخول التي عددها  $n$  مرتبة بحسب لحظات الانتهاء المتزايدة ببطاريات تبعاً للمعادلة (1.16)، وإلا فيمكننا فرضها بهذا الترتيب خلال زمن  $O(n \lg n)$  باختيار ترتيب عشوائي في حالة للسواة. في البدء، نضيف نشاطاً وهيئاً  $a_0$  حيث  $f_0 = 0$  بحيث تكون المسألة الجزئية  $S_0$  هي كامل مجموعة النشاطات  $S$ . الاستدعاء الابتدائي الذي يحل كامل المسألة هو **RECURSIVE-ACTIVITY-SELECTOR**( $s, f, 0, n$ ).

<sup>2</sup> لما كان شبه الرمز  $s$  يعبر  $s$  و  $f$  صفتين، فإنه يفهرس ضمنهما باستخدام أقواس مربعة عوضاً عن أدلة.

### RECURSIVE-ACTIVITY-SELECTOR( $s, f, k, n$ )

```

1   $m = k + 1$ 
2  while  $m \leq n$  and  $s[m] < f[k]$  // find the first activity in  $S_k$  to finish
3       $m = m + 1$ 
4  if  $m \leq n$ 
5      return  $\{a_m\} \cup \text{RECURSIVE-ACTIVITY-SELECTOR}(s, f, m, n)$ 
6  else return ■

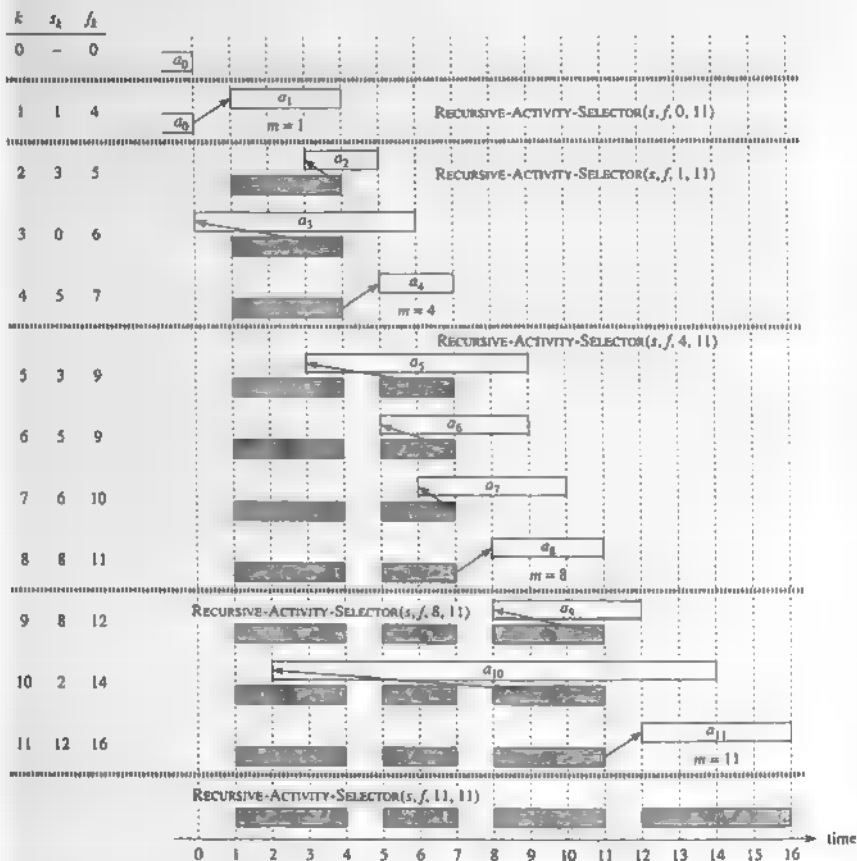
```

يبين الشكل 1.16 عمل هذه الخوارزمية. في أحد الاستدعاءات RECURSIVE-ACTIVITY-SELECTOR( $s, f, k, n$ )، تبحث حلقة **while** في السطرين 2-3 عن أول نشاط ينتهي في  $S_k$ . تفحص الحلقة  $a_{k+1}, a_{k+2}, \dots, a_n$  إلى أن تجد أول نشاط  $a_m$  متوافق مع  $a_k$  وهو نشاط يحقق  $s_m \geq f_k$ . إذا انتهت الحلقة - بسبب عثورها على مثل هذا النشاط - تعيد الإجراء في السطر 5 اجتماع  $\{a_m\}$  والمجموعة الجزئية ذات الحجم الأعظم لـ  $S_m$  التي أعادها الاستدعاء RECURSIVE-ACTIVITY-SELECTOR( $s, f, m, n$ ). وبالمقابل، يمكن أن تنتهي الحلقة لأن  $m > n$ ، وفي تلك الحالة تكون قد فحصنا كل النشاطات في  $S_k$  دون أن نجد النشاط المتوافق مع  $a_k$ . وفي هذه الحالة، يكون  $S_k = \emptyset$ ، وبذلك تعيد الإجراء ■ في السطر 6. بافتراض أن النشاطات كانت مفروزة تصاعديًا بحسب لحظات الانتهاء، يكون زمن تنفيذ الإجراء RECURSIVE-ACTIVITY-SELECTOR( $s, f, 0, n$ )،  $\Theta(n)$ ، ويمكن أن نرى ذلك كما يلي. عبر كل الاستدعاءات العُزديّة، يجري فحص كل نشاط مرة واحدة تمامًا في اختبار حلقة **while** في السطر 2. وبوجه خاص، يجري فحص النشاط  $a_i$  في آخر استدعاء كان فيه  $i < k$ .

### خوارزمية تكرارية شرهة

يمكننا بسهولة تحويل إجرائنا العُزدي إلى إجراء تكراري. إن الإجراء RECURSIVE-ACTIVITY-SELECTOR هو "عُزدي الذيل tail recursive" تقريبًا (انظر للسؤال 4-7): فهو ينتهي باستدعاء عُزدي له نفسه يتلوه عملية اجتماع. إن مهمة تحويل إجراء عُزدي الذيل إلى الشكل التكراري هي عادة مهمة مباشرة. في الحقيقة، تنجز بعض مترجمي لغات البرمجة هذه المهمة آليًا. يعمل الإجراء RECURSIVE-ACTIVITY-SELECTOR، كما هو مكتوب، على المسائل الجزئية  $S_k$ ، أي إن المسائل الجزئية تتكون من النشاطات التي تنتهي آخرًا (آخر النشاطات من حيث الانتهاء).

إن الإجراء GREEDY-ACTIVITY-SELECTOR هو نسخة تكرارية من الإجراء RECURSIVE-ACTIVITY-SELECTOR. وهو يفترض أيضًا أن نشاطات الدخول مرتبة بحسب لحظات الانتهاء لمتزايدة باطراد. فهو يجمع النشاطات المختارة في مجموعة  $A$ ، ويعيد هذه المجموعة حين ينتهي.



**الشكل 1.16** عمل الإجراء RECURSIVE-ACTIVITY-SELECTOR على 11 نشاطاً معطى سابقاً. تظهر النشاطات المدروسة لكل استدعاء بين الخطوط الأفقية. ينتهي النشاط الوهمي  $a_0$  في اللحظة 0، وفي الاستدعاء الأول  $\text{RECURSIVE-ACTIVITY-SELECTOR}(s, f, 0, 11)$ ، يجري اختيار النشاط  $a_2$ . نرى، لكل استدعاء غزوي، النشاطات التي جرى اختيارها سابقاً مظلة والنشاطات التي هي في قيد الدراسة بالأبيض. إذا كانت لحظة البدء لنشاط ما قبل لحظة انتهاء آخر نشاط مُضاف (السهم ينها يشير إلى اليسار)، يُستبعد هذا النشاط. وإلا (يشير السهم مباشرة إلى الأعلى أو إلى اليمين)، فيجري اختياره. الاستدعاء الغزوي الأخير  $\text{RECURSIVE-ACTIVITY-SELECTOR}(s, f, 11, 11)$  يعيد  $\emptyset$ . المجموعة الناتجة عن النشاطات هي  $\{a_1, a_2, a_3, a_{11}\}$ .

GREEDY-ACTIVITY-SELECTOR( $s, f$ )

```

1   $n = s.length$ 
2   $A = \{a_1\}$ 
3   $k = 1$ 
4  for  $m = 2$  to  $n$ 
5      if  $s[m] \geq f[k]$ 
6           $A = A \cup \{a_m\}$ 
7           $k = m$ 
8  return  $A$ 
```

يعمل الإجراء كما يلي: يشير المتحول  $k$  إلى أحدث إضافة إلى  $A$ ، الموافقة للنشاط  $a_k$  في النسعة القويّة. ولما كانت النشاطات مرتبة بحسب الترتيب المتزايد باطراد للحظات الانتهاء، فإن  $f_k$  هو دائماً لحظة الانتهاء المُعطى لأي نشاط في  $A$ . أي إن:

$$f_k = \max \{f_i : a_i \in A\}. \quad (3.16)$$

يختار السطران 2-3 النشاط  $a_1$ ، ثم يجري استبداء  $A$  لتتضمن هذا النشاط فقط، واستبداء  $k$  ليؤشر إلى هذا النشاط. ثم يُجد الحلقة **for** في الأسطر 4-7 النشاط الأكبر انتهاءً في  $S_k$ . تأخذ الحلقة كل نشاط  $a_m$  بالاعتبار، ونضيف  $a_m$  إلى  $A$  إذا كان متوافقاً مع جميع النشاطات المختارة سابقاً؛ مثل هذا النشاط هو أكبر نشاط ينتهي في  $S_k$ . ولمعرفة كون النشاط  $a_m$  متوافقاً مع جميع النشاطات الموجودة حالياً في  $A$ ، يكفي - اعتماداً على المعادلة (3.16) - فحص لحظة البدء  $s_m$  (السطر 5)، والتأكد أنها ليست أكبر من  $f_k$ ، زمن انتهاء آخر نشاط أُضيف إلى  $A$ . فإذا كان النشاط  $a_m$  متوافقاً، فإن السطرين 6-7 يضيفان هذا النشاط إلى  $A$  ويضعان القيمة  $m$  في  $k$ . إن المجموعة  $A$  التي أعادها الاستدعاء GREEDY-ACTIVITY-SELECTOR( $s, f, 0, n$ ) هي تماماً المجموعة التي أعادها الاستدعاء RECURSIVE-ACTIVITY-SELECTOR( $s, f, 0, n$ ).

يبدول الإجراء GREEDY-ACTIVITY-SELECTOR، شأنَ النسعة القويّة، مجموعةً مكونة من  $n$  نشاطاً خلال زمن  $\Theta(n)$ ، بافتراض أن النشاطات مفرّزة منذ البداية تبعاً للحظات انتهائها.

## تمارين

### 1-1.16

أعط خوارزمية برمجية ديناميكية لحلّ مسألة اختيار النشاطات، اعتماداً على العلاقة القويّة (2.16). اجعل خوارزمتك تحسب الحجم  $c[f, i]$  كما عُرّفت سابقاً، وتُنتج أيضاً المجموعة الجزئية ذات الحجم الأعظم من النشاطات المتوافقة فيما بينها. افترض أنه للدخلات مُرّزت كما في المعادلة (1.16). قارن زمن تنفيذ حلّك بزمن تنفيذ الخوارزمية GREEDY-ACTIVITY-SELECTOR.

### 2-1.16

افترض أننا عوضًا عن اختيارنا الدائم للنشاط الذي ينتهي أولاً، اخترنا آخر نشاط يبدأ ويكون متوافقًا مع جميع النشاطات المختارة سابقًا. بين كيف أن هذا النهج هو خوارزمية شرهة، وبرهن أنه يحقق حلًا أمثل.

### 3-1.16

إن أي نصح شره لمسألة اختيار النشاطات لن يُنتِج مجموعة ذات حجم أعظم من النشاطات المتوافقة فيما بينها. أعطِ مثالاً يبيّن أن النهج للتمكّل في اختيار النشاط الأقصر من تلك النشاطات المتوافقة مع النشاطات المختارة سابقًا لن يعمل. أعد الطلب نفسه للنهجين: الأول الذي يختار دائمًا النشاط المتوافق الذي يتداخل مع أقل عدد مع النشاطات المتبقية، والثاني الذي يختار دائمًا النشاط المتبقي المتوافق ذا أبكر لحظة بدء.

### 4-1.16

نفترض أن لدينا مجموعة نشاطات علينا جدولتها بين عدد كبير من قاعات المحاضرات، حيث يمكن أن يحصل أي نشاط في أية قاعة محاضرات. نوّد جدولّة جميع النشاطات باستخدام أقل عدد ممكن من قاعات المحاضرات. أعطِ خوارزمية شرهة فعالة لتحديد القاعة التي يستخدمها كل نشاط.

(تعرّف هذه المسألة أيضًا بمسألة **تلوين بيان-مجال interval-graph coloring problem**. يمكننا إنشاء بيان مجال عُقْدُهُ هي النشاطات للمعطة ووصلاته تربط النشاطات غير المتوافقة. إن أقل عدد من الألوان اللازمة لتلوين كل عقدة، بحيث لا نعطي لعقدتين متجاورتين اللون نفسه، يوافق إيجاد أقل عدد من قاعات المحاضرات لجدولة جميع النشاطات للمعطة.)

### 5-1.16

لندرس تعديلًا على مسألة اختيار النشاطات بحيث يكون لكل نشاط  $a_i$  قيمة  $v_i$  إضافة إلى لحظات البدء والانتهاء. ولم يُقدِّد الهدف جعل عدد النشاطات المجدولة أعظميًا، وإنما جعل القيمة الكلية للنشاطات المجدولة عظمى عوضًا عن ذلك. أي علينا اختيار مجموعة  $A$  من النشاطات المتوافقة بحيث يكون  $\sum_{a_i \in A} v_i$  أعظميًا. أعطِ خوارزمية تحلّ هذه المسألة بزمّن كثير حدودي.

## 2.16 عناصر الاستراتيجية الشرهة

توصّل الخوارزمية الشرهة إلى حلّ أمثل لمسألة ما باقتداء متتالية خيارات. لدى كل نقطة قرار في الخوارزمية، نختار الخيار الذي يبدو الأفضل عند تلك اللحظة. هذه الاستراتيجية الكسبية heuristic لا تُنتِج حلًا أمثلًا على الدوام، ولكن مثلما رأينا في مسألة اختيار النشاطات، يمكن أحيانًا أن تُنتِج حلًا أمثل. يناقش هذا المقطع بعض الخواص العامة للطرائق الشرهة.

لقد كانت الإجرائية التي اتبعناها في المقطع 1.16 لتطوير خوارزمية شرهة أكثر تعقيدًا بقليل من المعتاد.

لقد آتينا الخطوات التالية:

1. تعديل البنية الجزئية المثلى للمسألة.
2. تطوير حلٍّ عؤدي. (في مسألة اختيار النشاطات، قمنا بصياغة العلاقة العؤدية (2.16)، ولكننا لم نطور خوارزمية عؤدية تعتمد على تلك العلاقة.)
3. يأت أنه في حال اعتماد الخيار الشره، فإنه سيبقى مسألة جزئية وحيدة.
4. إثبات أن اعتماد الخيار الشره آمن دوماً (يمكن اعتماده دائماً). (يمكن أن تحدث الخطوات 3 و 4 بأي ترتيب.)
5. تطوير خوارزمية عؤدية تنجز الاستراتيجية الشرهة.
6. تحويل الخوارزمية العؤدية إلى خوارزمية تكرارية.

باتباع هذه الخطوات، رأينا بتفصيل كبير دعائم البرجة الديناميكية التي تركز عليها أية خوارزمية شرهة. على سبيل المثال، في مسألة اختيار النشاطات، عرفنا أولاً المسائل الجزئية  $S_r$ ، حيث كان كل من  $i$  و  $r$  يتغيران. ثم وجدنا لاحقاً أننا إذا اعتمدنا الخيار الشره دائماً، يمكن أن تقتصر مسائلنا الجزئية على الشكل  $S_k$ . وبالمقابل، كان يمكننا تشكيل البنى الجزئية المثلى على خلفية الخيار الشره، بحيث يترك الخيار مسألة جزئية واحدة لحلها. في مسألة اختيار النشاطات، كان بإمكاننا الاستغناء عن الدليل الثاني وتعريف المسائل الجزئية من الشكل  $S_k$ . ثم كان بإمكاننا برهان أنه بتراكب الخيار الشره (أول نشاط  $\alpha_m$  ينتهي في  $S_k$ )، مع حلٍّ أمثل لبقية المجموعة  $S_m$  من النشاطات المتوافقة، نحصل على حلٍّ أمثل لـ  $S_k$ . وبعمومية أكبر، نصمم الخوارزميات الشرهة باتباع الخطوات التالية:

1. تحويل مسألة الأمثلة إلى مسألة نتخذ فيها خياراً، ويبقى علينا حلُّ مسألة جزئية واحدة.
  2. برهان وجود حلٍّ أمثل للمسألة الأصلية دائماً، وهذا الحل يتخذ خياراً شرهاً، بحيث يكون اتخاذ الخيار الشره آمناً دائماً.
  3. إثبات البنى الجزئية المثلى ببيان أنه باتخاذ الخيار الشره فإن ما يتبقى هو مسألة جزئية تتمتع بالخاصية التالية: إذا راكبنا الحل الأمثل للمسألة الجزئية مع الخيار الشره نتوصل إلى حلٍّ أمثل للمسألة الأصلية.
- سنستخدم هذه الإحترافية المباشرة أكثر في مقاطع تالية من هذا الفصل. غير أنه يوجد في الغالب، خلف كل خوارزمية شرهة، حلٍّ أكثر تعقيداً يعتمد البرجة الديناميكية.
- كيف يمكننا القول بأن خوارزمية شرهة مستحل مسألة استمثال (أمثلة) خاصة؟ لا توجد طريقة واحدة تصلح لكل الحالات، غير أن خاصية الخيار الشره والبنى الجزئية المثلى هما المكونان الأساسيان لها. فإذا استطعنا إثبات أن للمسألة هاتين الخاصيتين، فإننا على طريق تطوير خوارزمية شرهة للحل.

### خاصية الخيار الشره

إن المكوّن الأساسي الأول هو خاصية الخيار الشره *greedy-choice property*: يمكننا تجميع حلّ شامل أمثل باعتماد خيار أمثل محلياً (شره). وبعبارة أخرى، حين نكون بصدد اعتماد أحد الخيارات، فإننا نعتمد الخيار الذي يبدو الأفضل في المسألة الحالية، دون الالتفات إلى نتائج المسائل الجزئية.

في هذه المرحلة، تختلف الخوارزميات الشرهة عن البرمجة الديناميكية. ففي البرمجة الديناميكية نعتمد خياراً لدى كل خطوة، إلا أن الخيار يعتمد عادة على حلول المسائل الجزئية. نتيجة لذلك، فإننا نحلّ عادةً مسائل البرمجة الديناميكية بطريقة صعودية، متقدّمين من مسائل جزئية صغرى إلى مسائل جزئية كبرى. (بالمقابل، يمكننا حلها نزولياً، ولكن باستدكار *memoizing*. ومع أن الرماز يعمل نزولياً، فما يزال علينا طبقاً حل المسائل الجزئية قبل اتخاذ الخيار). في الخوارزمية الشرهة نأخذ أي خيار يبدو الأفضل في تلك اللحظة، ثم نحلّ المسائل الجزئية المتبقية. يمكن أن يعتمد الخيار الذي نأخذه في الخوارزمية الشرهة على الخيارات الماضية، ولكنه لا يمكن أن يعتمد على أي خيارات مستقبلية أو على حلول للمسائل الجزئية. وهكذا، وعلافاً للبرمجة الديناميكية، التي تحلّ المسائل الجزئية قبل اتخاذ الخيار الأول، فإن الخوارزمية الشرهة تتخذ خيارها الأول قبل حل أية مسألة جزئية. تتقدم خوارزمية البرمجة الديناميكية صعودياً، في حين تتقدم الاستراتيجية الشرهة عادةً نزولياً، بأخذ خيار شره واحد بعد الآخر، بحيث نقصص كل متنسخ *instance* للمسألة إلى مسألة أصغر منها. علينا بالطبع أن نثبت أن الخيار الشره يعطي حلاً شاملاً أمثل، عند كل خطوة. وكما هو الحال في المرحلة 1.16، فإن البرهان يدرس عادةً حلاً شاملاً أمثل لمسألة جزئية. ثم يبين كيفية تعديل الحل للاستعاضة عن الخيار الشره بخيارات أخرى ينتج عنها مسائل جزئية مشابهة، ولكن أصغر من المسألة الأساسية.

يمكننا عادةً اتخاذ الخيار الشره بفعالية أكثر من حالة اتخاذ مجموعة أوسع من الخيارات. ففي مسألة اختيار النشاطات مثلاً، احتجنا لفحص كل نشاط مرة واحدة فقط، وذلك بفرض أننا فرزنا سلقاً النشاطات بحسب الترتيب المتزايد باطراد للحفظات الانتهاء. بمعالجة بدائية للدخل أو باستخدام بنية معطيات مناسبة (وهي غالباً رتل ذو أولوية)، يمكننا أخذ خيارات شرهة بسرعة، وتوصّل بذلك إلى خوارزمية فعّالة.

### بنية جزئية مُثلى

تُبدى مسألة ما بنية جزئية مُثلى *optimal substructure* إذا تضمنت الحل الأمثل للمسألة حلولاً مُثلى لمسائل جزئية. هذه الخاصية هي إحدى المكونات الأساسية لتقدير قابلية تطبيق البرمجة الديناميكية أو الخوارزميات الشرهة. كمثال على البنى الجزئية المُثلى، نذكر كيف رهناً في المقطع 1.16، أنه إذا تضمنت حلّ أمثل لمسألة جزئية  $S_i$  نشاطاً  $a_k$ ، وجب عندها أن يتضمن أيضاً حلولاً مُثلى للمسائل الجزئية  $S_{i-k}$  و  $S_{i-k+1}$ . واعتماداً على هذه البنية الجزئية المُثلى، ناقشنا أنه إذا علمنا أي نشاط علينا استعماله على أنه  $a_k$ ، أمكننا بناء حلّ أمثل للمسألة  $S_i$  باختيار النشاط  $a_k$  مع جميع النشاطات في حلول مُثلى للمسائل



الجزئية  $z_k$ ، و  $z_k$ . وبالاعتماد على هذه للملاحظة التي تخص البنى الجزئية المثلى، أمكننا استنباط العلاقة القويّة (2.16) التي وصفت قيمة حلّ أمثل.

نستخدم عادةً نمطاً أشدّ وضوحاً فيما يخص البنى الجزئية المثلى حين تطبيقها على الخوارزميات الشرية. وكما ذكرنا آنفاً، أفردنا في افتراض أننا وصلنا إلى مسألة جزئية (أي من الصيغة نفسها)، باتخاذ خيار شره في المسألة الأصلية، على حين أن كل ما يلزمنا حقيقة هو أن نناقش إذا كان ضمن الحل الأمثل للمسألة الجزئية إلى الخيار الشره الذي المتخذ، يعطي حلاً أمثل للمسألة الأصلية. يُستخدم هذا المنهج ضمناً الاستقراء على المسائل الجزئية لإثبات أن اتخاذ الخيار الشره عند كل خطوة ينتج حلاً أمثل.

### الخيار الشره مقابل البرمجة الديناميكية

لما كانت خاصية البنى الجزئية المثلى مستخدمة في كلٍّ من الاستراتيجيات الشرية والبرمجة الديناميكية، فقد ترغب في توليد حلّ بالبرمجة الديناميكية لمسألة ما عندما يكون الحلّ الشره كافياً، أو بالعكس، تفكر خطأ في أن الحلّ الشره ناجح في حين يتطلب الأمر في الحقيقة حلاً بالبرمجة الديناميكية. وليبيان الفوارق الدقيقة بين التقنيتين، سنتفحص نوعين مختلفين لمسألة أمثلة معروفة.

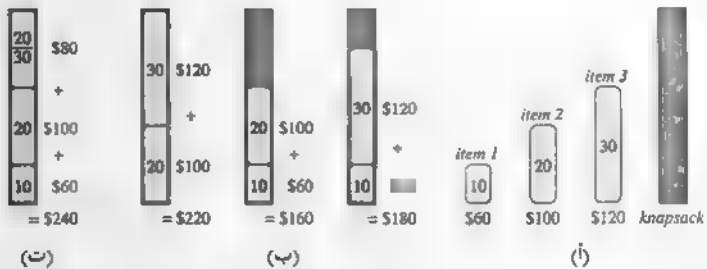
**مسألة حقيبة الظهر 0-1 knapsack problem** هي التالية: بسرقة لصٌ مخزوناً فيجد فيه  $n$  غرضاً؛ فبما أن الغرض  $i$  له وزن  $w_i$  ودولاراً  $v_i$  و  $w_i$  رطلاً، حيث  $v_i$  و  $w_i$  أعداد صحيحة. يريد اللص أن يأخذ الحِمل الأعلى ممّا قدّر الإمكان، إلا أن بإمكانه حمل  $W$  رطلاً على الأكثر في حقيبة ظهره حيث  $W$  عدد صحيح ما. ما هي الأغراض التي ينبغي أن يأخذها؟ (سمّيت هذه المسألة مسألة حقيبة الظهر 0-1 لأن أيّ غرض من الأغراض إما أن يأخذه اللصّ وإما يدعه جانباً ولا يستطيع اللصّ أخذ جزء من غرض، أو يأخذ غرض أكثر من مرة واحدة.)

أما في مسألة حقيبة الظهر الكسرية *fractional knapsack problem*، فللمشهد هو نفسه، غير أن اللصّ يستطيع أخذ كسور الأغراض، بدلاً من أن يكون لديه الخيار (0-1) لكل غرض. ويمكنك أن تتخيل الأغراض في مسألة حقيبة الظهر 0-1 على أنها سبائك ذهبية، أما الأغراض في مسألة حقيبة الظهر الكسرية فهي أكثر شبهها بشلور الذهب.

تتمتع كلٌّ من مسألتَي حقيبة الظهر بخاصية البنى الجزئية المثلى. لننرس، في المسألة 0-1، الحِمل الأمثل الذي وزنه  $W$  رطلاً على الأكثر. إذا حذفنا الغرض  $z$  من الحِمل فإن الحِمل المتبقي هو الحِمل الأمثل الذي يزن  $W - w_z$  على الأكثر، الذي يستطيع اللصّ أخذه من  $1 - z$  غرضاً أصلياً باستبعاد  $z$ . وفي المسألة الكسرية المشابهة، نعتبر أننا إذا حذفنا وزناً  $w_z$  من أحد العناصر  $z$  من الحِمل الأمثل، ونحب أن يكون الحِمل المتبقي هو الحِمل الأمثل ذا الوزن  $W - w_z$  على الأكثر، الذي يستطيع اللصّ أخذه من الأغراض الأصلية  $1 - z$  إضافة إلى  $w_z - w_z$  رطلاً من الغرض  $z$ .

ومع أن المسائلتين متشابهتان، فيمكننا حل مسألة حقيبة الظهر الكسرية باستخدام استراتيجية شرهة، ولكننا لا نستطيع حل المسألة 0-1 بالاستراتيجية نفسها. ولحل المسألة الكسرية، نحسب أولاً قيمة الرطل الواحد  $v_i/w_i$  لكل غرض. وباتباع استراتيجية شرهة، يبدأ اللص بأخذ ما يمكن من الغرض ذي القيمة العليا للرطل الواحد. فإذا نفد ذلك الغرض، وكان بإمكان اللص أن يحمل أكثر، فإنه يأخذ ما يمكنه أخذه من الغرض ذي القيمة العليا التالية للرطل الواحد، وهكذا، إلى أن لا يعود بإمكانه حمل أي شيء زائد. وهكذا، وبغرض الأغراض بحسب قيمة الرطل الواحد لكل منها، تُنفَّذ الخوارزمية الشرهة بزمن  $O(n \log n)$ . وسيُطلب إليك في التمرين 2.16-1 البرهان على أن مسألة حقيبة الظهر الكسرية تتمتع بخاصية الخيار الشره.

ولعرفة أن الاستراتيجية الشرهة لا تعمل في مسألة حقيبة الظهر 0-1، نأخذ منتج instance للمسألة المبين في الشكل 2.16 (أ). ففي هذا المثال ثلاثة عناصر، وبإمكان حقيبة الظهر حمل 50 رطلاً. العنصر 1 وزن 10 أرطال وقيمه 60 دولاراً، والعنصر 2 وزن 20 رطلاً وقيمه 100 دولاراً، والعنصر 3 وزن 30 رطلاً وقيمه 120 دولاراً. وهكنا، فإن قيمة كل رطل من العنصر 1 تساوي 6 دولاراً، وهي أكبر من قيمة الرطل لأي من العنصر 2 (5 دولارات لكل رطل) أو العنصر 3 (4 دولارات لكل رطل). لذلك، فإن الاستراتيجية الشرهة ستأخذ العنصر 1 أولاً. ولكن، كما يظهر من تحليل الحالة في الشكل 2.16 (ب)، فإن الحل الأمثل يأخذ العنصرين 2 و 3 تاركاً العنصر 1. أما الحلول للمكان الآخرون اللذان يأخذان العنصر 1 بالاعتبار فهما غير أمثلين.



**الشكل 2.16** مثال يبيّن أن الاستراتيجية الشرهة لا تناسب حالة مسألة حقيبة الظهر 0-1. (أ) على اللص أن يختار مجموعة جزئية من الأغراض الثلاثة المبينة بحيث لا يزيد وزنها الكلي عن 50 رطلاً. (ب) تتضمن المجموعة الجزئية المثلى الغرضين 2 و 3. وأي حل يتضمن الغرض 1 هو حل غير أمثل (أمثل جزئياً)، بالرغم من أن للغرض 1 أعلى قيمة للرطل الواحد. (ت) في حالة مسألة حقيبة الظهر الكسرية، فإن أخذ الأغراض بحسب ترتيب أعلى قيمة لكل رطل يعطي حلاً أمثل.

ولكن، في المسألة الكسرية للشاشمة، تؤدي الاستراتيجية الشرمة التي تأخذ العنصر 1 أولاً إلى حل أمثل، كما هو مبين في الشكل 2.16(ت). فيما لا يمكن أخذ العنصر 1 في المسألة 0-1 لأن ذلك يمنع اللص من ملء حقيبة ظهره بكل سعتها، والفرغ في الحقيقة يخفّض القيمة الفعلية للربط الواحد من حمله. حين تأخذ بالاعتبار عنصراً لتضمينه في الحقيبة الظهريّة في المسألة 0-1، علينا أن نقارن حلّ للمسألة الجزئية الذي يضمّن هذا العنصر بحلّ للمسألة الجزئية الذي يستبعد هذا العنصر قبل اعتماد خيارنا. ينتج عن صياغة المسألة بهذه الطريقة، مسائل جزئية كثيرة متراكبة، وهذه سمّة مميزة للبرمجة الديناميكية، وبالفعل، مثلما يُطلب إليك بيانه في التمرين 2-2.16، يمكن استخدام البرمجة الديناميكية لحلّ للمسألة 0-1.

### تعاريف

#### 1-2.16

برهن أن مسألة حقيبة الظهر الكسرية تتمتع بخاصية الخيار الشرمة.

#### 2-2.16

أعط حلّاً بالبرمجة الديناميكية لمسألة حقيبة الظهر 0-1 تُنفَّذ بزمن  $O(nW)$ ، حيث  $n$  عدد العناصر، و  $W$  الوزن الأعظم للأغراض التي يستطيع اللص وضعها في حقيبة ظهره.

#### 3-2.16

افترض في مسألة حقيبة الظهر 0-1 أن ترتيب الأغراض حين فرزها تصاعدياً تبعاً للوزن، هو نفسه ترتيبها بحسب قيمها المئناقصية. أعط خوارزمية فعالة لإيجاد الحل الأمثل لهذا الشكل المعدّل من مسألة حقيبة الظهر، وناقش صحة خوارزمتك.

#### 4-2.16

يُعلم البروفسور جيكو Gekko دائماً بعبور داكوتا الشمالية North Dakota على المزلّاج، وهو يُخطّط لعبور الولاية على الطريق السريع U.S. 2، الذي يبدأ من Grand Forks، على الحدود الشرقية مع مينيسوتا Minnesota، إلى ولستون Williston، قرب الحدود الغربية مع مونتانا Montana. يمكن أن يحمل البروفسور لترتّن من الماء، ويمكنه أن يتربّح  $m$  ميلاً قبل أن يتفد ماؤه. (لأن داكوتا الشمالية سهليّة نسبياً، ليس على البروفسور أن يعبأ بشربه لماء بمعدلات أعلى عند للمقاطع الصاعدة منها عند المقاطع السهليّة أو الهابطة.) سيبدأ البروفسور عند Grand Forks بترتّي ماء كاملين. تبيّن خارطته الرسمية لولاية داكوتا الشمالية جميع الأماكن على طول الطريق U.S. 2 التي يمكنه عندها إعادة ملء الماء، والمسافات بين هذه المواضع.

هدف البروفسور هو تصغير عدد التوقيفات المائية على طول الطريق عبر الولاية. أعط طريقة فعّالة يستطيع بواسطتها تحديد التوقيفات التي يجب أن يقوم بها لملء الماء. بيّن أن استراتيجيتك تحقّق حلّاً أمثل، وأعط زمن التنفيذ.

### 5-2.16

صِفْ خوارزمية فعالة، تأخذ مجموعة من النقاط  $\{x_1, x_2, \dots, x_n\}$  على المستقيم الحقيقي، وتحدد أصغر مجموعة من المجالات المغلقة التي طولها واحد تحتوي كل النقاط للمعطاء. ناقش صحة خوارزمتك.

### \* 6-2.16

بَيِّن كيف تحل مسألة حفيية الظهر الكسرية بزمن  $O(n)$ .

### 7-2.16

افترض أنَّ لديك مجموعتين  $A$  و  $B$ ، تتضمن كلٌّ منها  $n$  عددًا صحيحًا موجبًا. يمكنك إعادة ترتيب كل مجموعة بالطريقة التي تريد. بعد إعادة الترتيب، ليكن  $a_i$  العنصر ذا الترتيب  $i$  من المجموعة  $A$ ، و  $b_i$  العنصر ذا العنصر  $i$  من المجموعة  $B$ . بعد ذلك ستحصل على مكافأة بقيمة  $\prod_{i=1}^n a_i b_i$ . أعطِ خوارزمية تجعل المكافأة عظمى، أثبت أن خوارزمتك تجعل المكافأة عظمى، وصرِّح عن زمن التنفيذ.

## 3.16 أرمزة هوفمان

تضغط أرمزة هوفمان المعطيات بفعالية عالية جدًا ويمكن عادة تحقيق وفرة بنسبة 20% إلى 90%، وذلك تبعًا لخواص المعطيات التي هي في قيد الضغط. نفترض أن المعطيات هي متتالية من الحروف. نستخدم خوارزمية هوفمان الشجرة جدولاً بمزات ورود الحروف (أي تواتراتها) لبناء طريقة مثلى لتمثيل كل حرف كسلسلة حروف الثنائية.

افترض أن لدينا ملف معطيات بـ 100,000 حرف، ونود تخزينه تخزينًا مفاضًا. نلاحظ أن الحروف في الملف تُرد بالتواترات للمعطاء بالشكل 3.16. أي تظهر فقط 6 حروف مختلفة، ويُرد الحرف  $a$  45,000 مرة. ثمة عدة خيارات لتمثيل ملف معلومات كهذا. هنا، نختم بمسألة تصميم رماز بالمحارف الثنائية *binary character code* (أو باختصار رماز *code*) حيث نركز كل حرف بمتتالية حروف ثنائية وحيدة. إذا استخدمنا رمازًا *معدّد الطول fixed-length code* فإننا نحتاج إلى ثلاثة بتات لتمثيل الحروف الستة:  $a = 000, b = 001, \dots, f = 101$ . تتطلب هذه الطريقة 300,000 بتًا لتمييز كامل الملف. هل يمكننا القيام بما هو أفضل من هذا؟

يمكن للرماز المتغير الطول *variable-length code* أن يكون أدائه أفضل بكثير من الرماز المحدد الطول، بإعطاء الحروف العالية التواتر كلمات رماز قصيرة والحروف النادرة كلمات رماز طويلة. يبيّن الشكل 3.16 هذا الرماز؛ هنا، يمثل الحرف ذو البت الوحيد 0 الحرف  $a$ ، ويمثل الحرف ذي البتات الأربعة 1100 الحرف  $f$ . يتطلب هذا الرماز:

$$\text{بت } 224,000 = (1,000) + [(5)(4)] + (9)(4) + (16)(3) + (12)(3) + (13)(3) + (1)(45)$$

المحرف	a	b	c	d	e	f
نواتره (بالآلاف)	45	13	12	16	9	5
كلمة الرمز بطول ثابت	000	001	010	011	100	101
كلمة الرمز بطول متغير	0	101	100	111	1101	1100

**الشكل 3.16** مسألة ترميز محارف. يتضمن ملف معطيات 100,000 محرفًا، من مجموعة المحارف  $a-z$  فقط بالتواترات المبينة. إذا أسندنا رمازًا من ثلاثة بتات لكل محرف، فيمكن ترميز الملف بـ 300,000 بت. أما باستخدام الرمز المتغير الطول المبين، فيمكن ترميز الملف بـ 224,000 بت.

لتمثيل الملف، أي بوفر نسبة 25% تقريبًا. وهو في الواقع رماز أمثل للمحارف لهذا الملف، مثلما سنرى لاحقًا.

### الأرمزة السبقية

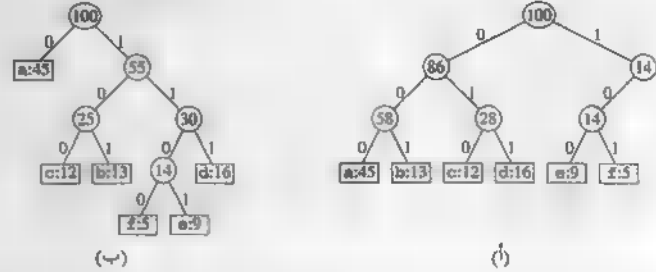
سندرس هنا فقط الأرمزة التي لا تكون فيها أية كلمة رماز سابقة prefix لكلمة رماز أخرى. تسمى هذه الأرمزة **أرمزة سبقية** <sup>3</sup> *Prefix codes*. يمكن أن نبيّن، علمًا بأننا لن نبرهن ذلك هنا، أن الرمز السبقية يمكنه دائمًا إنجاز ضغط المعطيات الأمثل من بين أي رماز للمحارف، وبذلك فإننا لا نقص من عمومية المسألة إذا قصرنا اهتمامنا على الأرمزة السبقية.

إن عملية الترميز بسيطة لأي رماز محارف ثنائية؛ إذ يكفي ضم كلمات الرمز التي تمثل كل محرف في الملف. على سبيل المثال، نرّمز ملف المحارف الثلاثة abc بالرمز المتغير الطول السبقية الموضح بالشكل 3.16 كما يلي:  $0101100 = 0 \cdot 101 \cdot 100$ ، حيث يشير "0" إلى الضم.

الأرمزة السبقية مرغوبة لأنه يمكن فك ترميزها ببساطة. ولما كانت أية كلمة رماز لا تكون سابقة لأية كلمة رماز أخرى، فإن كلمة الرمز التي يبدأ بها ملف مرّمز غير مُلبّسة. يمكننا ببساطة تحديد كلمة الرمز الأولى، وإعادةها إلى المحرف الأصلي، وتكرار عملية فك ترميز بقية الملف المرّمز. في مثالنا، نحلّل سلسلة المحارف 001011101 تحليلًا وحيثًا إلى 001011101، ونفكّ ترميزها إلى aabe.

تحتاج عملية فك الترميز إلى تمثيل مناسب للأرمزة السبقية، بحيث يمكننا التقاط كلمة الرمز الأولى بسهولة. توفر الشجرة الثنائية التي أوفاقها هي المحارف المعطاة مثل هذا التمثيل. نفسر الكلمة الثنائية محرف على أنها المسار البسيط من الجذر إلى ذلك المحرف، بحيث يعني 0 "اذهب إلى الابن الأيسر" و 1 "اذهب إلى الابن الأيمن". يبيّن الشكل 4.16 أشجار الرمازين في مثالنا. لاحظ أن هذه الأشجار ليست أشجار بحث ثنائية، لأنها لا تتطلب ظهور الأوراق بترتيب مفروز، ولأن العقد الداخلية لا تتضمن مفاتيح محرفية.

<sup>3</sup> ربما تكون التسمية "أرمزة من دون سوابق" أفضل، إلا أن "الأرمزة السبقية" معيارية في المراجع.



**الشكل 4.16** الأشجار الموافقة لأساليب الترميز في الشكل 3.16. كل ورقة لها لصيقة بالمحرف مع تواتر ورودها، وكل عقدة داخلية لها لصيقة بمجموع تواتر ورود الأوراق في شجرة الفرعية. (أ) الشجرة الموافقة للرماز المحدد الطول  $a = 000, \dots, f = 101$ . (ب) الشجرة الموافقة للرماز السبقى الأمل  $a = 000, \dots, f = 101$ .

يجري دائما تغيل الرماز الأمل لطف بشجرة ثنائية مملأ *full*، بحيث يكون لكل عقدة إن لم تكن ورقة ابنان (انظر التمرين 2-3.16). إن الرماز المحدد الطول في مثالنا ليس أمثلًا، لأن شجرته المبينة في الشكل 4.16 (أ) ليست شجرة ثنائية مملأ: ثمة كلمات رماز تبدأ بـ 10... ولكن لا تبدأ أي كلمة رماز بـ 11... ولأننا سنقصر اهتمامنا الآن على الأشجار الثنائية المملأ، يمكننا القول إنه إذا كانت  $C$  الأبجدية التي تنتمي إليها المحارف، وكانت تواترات جميع المحارف موجبة، كان لأشجار الأرمزة السبقية المثلى  $|C|$  ورقة بالضبط، ورقة لكل حرف في الأبجدية، وكان لها تمامًا  $|C| - 1$  عقدة داخلية (انظر التمرين 3-5.ب).

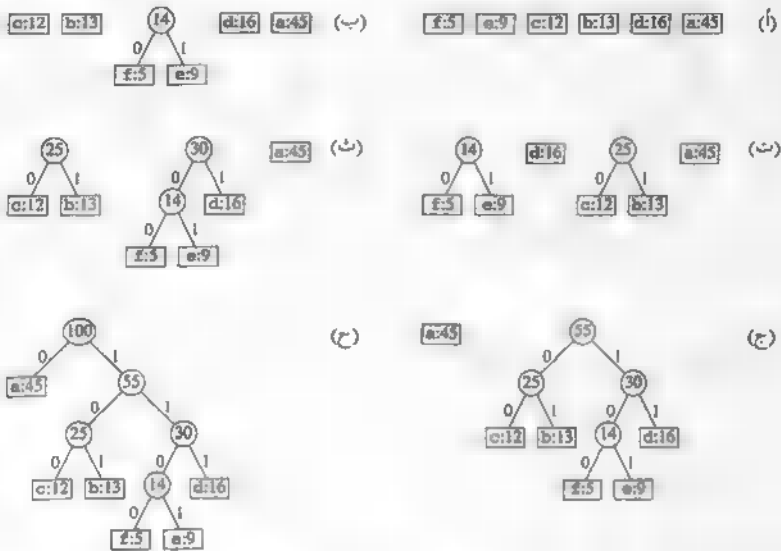
إذا كان لدينا شجرة  $T$  توافقي رمازًا سبقيًا، يمكن بسهولة عدّ البتات اللازمة لترميز الملف. ل نرمز بالوصفة  $c.freq$  إلى تواتر الحرف  $c$  من الأبجدية  $C$  في الملف، وليكن  $d_T(c)$  عمق ورقة الحرف  $c$  في الشجرة. لاحظ أن  $d_T(c)$  هو أيضًا طول كلمة ترميز الحرف  $c$ . وبذلك، يكون عدد البتات التي تتطلبها ترميز الملف هو:

$$B(T) = \sum_{c \in C} c.freq.d_T(c) \quad (4.16)$$

ونعرف هذا العدد بأنه **كلفة** *cost* الشجرة  $T$ .

**بناء رماز هوفمان**

اخترع هوفمان خوارزمية شرهة نبي رمازًا سبقيًا أمثل يسمى **رماز هوفمان** *Huffman code*. ونماشيا مع ملاحظتنا في المقطع 2.16، تعتمد صحة هذا الرماز على خاصية اختيار الشره والبنى الجزئية المثلى. وعوضًا عن تبيان تحقق هذه الخواص ثم تطوير شبه الرماز، فإننا سنعرض شبه الرماز أولاً. إذ سيساعد هذا في توضيح كيفية قيام الخوارزمية بالخيارات الشرهة.



**الشكل 3.16** خطوات خوارزمية هوفمان للتواترات المغطاة في الشكل 3.16. يبين كل جزء محتويات الرتل مفروزة تصاعدياً تبعاً للتواتر. (ي) كل خطوة، تُدمج الوثقتان الأقل تواتراً. تظهر الأوراق على شكل مستطيلات تحتوي الحرف وتواتره، والفُقد الداخلي على شكل دوائر تتضمن مجموع تواترات أبنائها. توضع لصاقة على الوصلة بين العقد الداخلية وأبنائها، فيسُها 0 إذا كانت الوصلة إلى اليمين الأيسر، و 1 إذا كانت الوصلة إلى اليمين الأيمن. إن كلمة رماز عقد، هي متتالية اللصاقات على الوصلات التي تربط الجذر بورقة ذلك الحرف. (أ) المجموعة البدائية من  $n = 1$  عقد، عقدة لكل حرف. (ب)-(ج) مراحل وسيطة. (ح) الشجرة النهائية.

نفترض في شبه الرماز التالي أن  $C$  مجموعة من  $n$  حرفاً، وأن كل حرف  $c \in C$  هو غرض له واصفة تعطي تواتره  $c.freq$ . تبني الخوارزمية الشجرة  $T$  الموافقة للرماز الأمثل بطريقة صعودية. فهي تبدأ بمجموعة من  $|C|$  ورقة، ثم تنحز متتالية من  $|C| - 1$  عملية "دمج" لإنشاء الشجرة النهائية. تستخدم الخوارزمية رتلاً ذا أولوية الأصغر  $Q$ ، مفتاحه الواصفة  $freq$ ، لتعيين الغرضين ذوي التواتر الأقل لدمجهما. إن ناتج الدمج هو غرض جديد تواتره هو مجموع تواتري الغرضين اللذين دُججا.

HUFFMAN( $C$ )

```

1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $z.left = x = \text{EXTRACT-MIN}(Q)$ 
```

```

6  z.right = y = EXTRACT-MIN(Q)
7  z.freq = x.freq + y.freq
8  INSERT(Q, z)
9  return EXTRACT-MIN(Q)    // return the root of the tree

```

تعمل خوارزمية هوفمان في مثالنا كما هو مبين في الشكل 5.16. ولما كانت الأبجدية تتضمن 6 محارف، فإن حجم الرتل البدائي  $n = 6$ ، ويلزم 5 خطوات دمج لبناء الشجرة. تمثل الشجرة النهائية الرماز السبق الأمثل. إن كلمة الرماز مخوف هي متتالية لصيقات الوصلات على المسار البسيط من الجذر إلى الحرف. يستبدئ السطر 2 رتل ذو أولوية الأصغر  $Q$  بملته محارف  $C$ . تستخرج حلقة **for** في الأسطر 3-8، تكرارًا، العقدة  $x$  و  $y$  الأدنى تواترًا في الرتل، وتستعيض عنهما في الرتل بعقدة جديدة  $z$  تمثل ناتج دمجهما. يُحسب تواتر  $z$  في السطر 7 على أنه مجموع تواتري  $x$  و  $y$ . للعقدة ■ ابنان، ■ هو الابن الأيسر و  $y$  هو الابن الأيمن. (هذا الترتيب اعتباطي؛ فالتبديل بين الابن الأيسر والأيمن لأية عقدة يُنتج رمازًا مختلفًا ولكن له الكلفة نفسها). بعد  $n - 1$  عملية دمج، يعيد السطر 9 العقدة الوحيدة المتبقية في الرتل، التي هي جذر شجرة الرماز.

ومع أن الخوارزمية ستعطي نفس النتيجة لو أننا أزلنا المتحولين  $x$  و  $y$  (بإسناد القيم مباشرة إلى  $z.left$  و  $z.right$  في السطرين 5 و 6، وتغير السطر 7 إلى  $z.freq = z.left.freq + z.right.freq$ )، فإننا سنستخدم الاسمين  $x$  و  $y$  لبرهان صحة الخوارزمية. لذلك، رأينا من المناسب الإبقاء عليهما. لتحليل زمن تنفيذ خوارزمية هوفمان نفترض أن الرتل  $Q$  منجز ككومة اثنائية وفق الأصغر binary min-heap (انظر الفصل 6). في حال تكونت المجموعة  $C$  من  $n$  حرفًا، يمكننا إنجاز استبداء الرتل  $Q$  في السطر 2 بزمن  $O(n)$  باستخدام الإجراء BUILD-MIN-HEAP الذي ناقشناه في اللقطة 3.6. نُنفذ حلقة **for** في الأسطر 3-8 مرةً تمامًا. ولما كانت كلُّ عملية كومة تتطلب زمنًا  $O(\lg n)$ ، فإن الحلقة تساهم بزمن  $O(n \lg n)$  في زمن التنفيذ. وبذلك يكون زمن التنفيذ الكلي لـ HUFFMAN على مجموعة من  $n$  حرفًا هو  $O(n \lg n)$ . يمكننا خفض زمن التنفيذ إلى  $O(n \lg \lg n)$  بالاستعاضة عن الكومة الثنائية وفق الأصغر بشجرة van Emde Boas (انظر الفصل 20).

### صحة خوارزمية هوفمان

لإثبات أن خوارزمية HUFFMAN الشجرة صحيحة، سنبيّن أن مسألة تعيين رماز سبقي أمثل تحقق خاصيتي الخيار الشره والبنية الجبرئية المثلى. تبين التوطئة التالية أن خاصية الخيار الشره عميقة.

#### توطئة 2.16

إذا كانت  $C$  أبجدية بحيث يكون لكل حرف  $c \in C$  تواتر  $c.freq$ ، وإذا كان  $x$  و  $y$  حرفين في  $C$  لهما أصغر



التواترات، فيوجد رمازٌ سبقيٌّ أمثل  $C$  optimal prefix code بحيث يكون لكل رمزي الخاصتين  $x$  و  $y$  الطول نفسه، وتختلفان في البت الأخير فقط.

**البرهان** تكمن فكرة البرهان في أخذ الشجرة  $T$  التي تمثل رمازًا سبقيًّا أمثلًا اعتباطيًا، وتعديلها لبناء شجرة تمثل رمازًا سبقيًّا أمثلًا آخر بحيث يظهر الحرفان  $x$  و  $y$  كورقتين أخنتين لهما أعظم عمق في الشجرة الجديدة. إذا استطعنا بناء مثل هذه الشجرة، فسيكون لكل رمزي الخاصتين  $x$  و  $y$  الطول نفسه، وستختلفان في البت الأخير.

ليكن الحرفان  $a$  و  $b$  ورقتين أخنتين لهما العمق الأعظم في  $T$ . ولكي لا نفقد عمومية الحل، نفترض أن  $a.freq \leq b.freq$  و  $x.freq \leq y.freq$ . ولما كان  $x.freq$  و  $y.freq$  أصغر تواترين مرتبين للأوراق، وكان  $a.freq$  و  $b.freq$  تواترتين اعتباطيتين مرتبين، فإن  $x.freq \leq a.freq$  و  $y.freq \leq b.freq$ . ومع ذلك، في بقية البرهان، من الممكن أن يكون لدينا  $x.freq = a.freq$  أو  $y.freq = b.freq$ . وإذا كان  $x.freq = b.freq$  فسيكون لدينا أيضًا  $x.freq = y.freq = a.freq = b.freq$  (انظر التمرين 1-3.16)، وستكون التوطئة مرفقة بدايةً. لذلك سنفترض أن  $x.freq \neq b.freq$ ، وهذا يعني أن  $x \neq b$ .

كما يبين الشكل 6.16، نبادل موضعي  $a$  في  $T$  لإنتاج شجرة  $T'$ ، ثم نبادل موضعي  $b$  و  $y$  في  $T'$  لإنتاج شجرة  $T''$  تكون فيها الورقتان  $x$  و  $y$  أخنتين بعمق أعظم. (لاحظ أنه إذا كان  $x = b$  ولكن  $y \neq a$ ، عندها لن تكون  $x$  و  $y$  أخنتين في الشجرة  $T''$  بعمق أعظم. ولأننا نفترض أن  $x \neq b$  فإن هذه الحالة لن تحصل.) من المعادلة (4.16) يكون فرق التكلفة بين  $T$  و  $T'$  هو:

$$\begin{aligned} B(T) - B(T') &= \sum_{c \in C} c.freq.d_T(c) - \sum_{c \in C} c.freq.d_{T'}(c) \\ &= x.freq.d_T(x) + a.freq.d_T(a) - x.freq.d_{T'}(x) - a.freq.d_{T'}(a) \\ &= x.freq.d_T(x) + a.freq.d_T(a) - x.freq.d_T(a) - a.freq.d_T(x) \\ &= (a.freq - x.freq)(d_T(a) - d_T(x)) \\ &\geq 0. \end{aligned}$$

لأن كلاً من  $a.freq - x.freq$  و  $d_T(a) - d_T(x)$  غير سالب. وبتخصيص أدق،  $a.freq - x.freq$  غير سالب لأن  $a$  هي ورقة ذات أصغر تواتر، و  $d_T(a) - d_T(x)$  غير سالب لأن  $a$  ورقة ذات عمق أعظم في  $T$ . وبالمثل، فإن المبادلة بين  $y$  و  $b$  لا يزيد التكلفة، وبذلك يكون  $B(T') - B(T'') \geq 0$ ، ولذلك، فإن  $B(T'') \leq B(T)$ ، ولما كانت  $T$  مثلى، فإن  $B(T'') = B(T)$ ، وهذا يقضي أن  $B(T'') = B(T)$ . إذن،  $T''$  شجرة مثلى تظهر فيها  $x$  و  $y$  كورقتين أخنتين لهما عمق أعظم، ومنه نستنتج صحة التوطئة. ■



**الشكل 6.16** توضيح الخطوة المفتاحية في برهان التوطئة 2.16. في الشجرة المثلى  $T$ ، الورقتان  $a$  و  $b$  هما أختان تتمتعان بمسعى أعظم. الورقتان  $x$  و  $y$  هما أخرفان اللذان لهما التواتر الأقل؛ تظهران بموضعين اعتباطيين في الشجرة  $T$ . بافتراض أن  $x \neq b$ ، فإن إبدال الورقتين  $x$  و  $b$  فيما بينهما يُنتج الشجرة  $T''$ ، ثم يُنتج إبدال الورقتين  $b$  و  $y$  الشجرة  $T'''$ . ولما كانت أية عملية إبدال لا تزيد في الكلفة، فإن الشجرة الناتجة  $T'''$  هي أيضاً شجرة مثلى.

تتضمن التوطئة 2.16 أنه يمكن، دون المساس بعمومية المسألة، بدءاً إجرائية بناء شجرة مثلى باستخدام الخيار الشرة الذي يدمج الأخرفين ذوي التواتر الأقل. لماذا هذا الخيار هو خيار شره؟ يمكننا رؤية تكلفة عملية دمج وحيد على أنها مجموع تواتري العنصرين للدموجين. يتبين الصريح 3.16-4 أن التكلفة الكلية للشجرة التي جرى بناؤها يساوي مجموع تكلفة عمليات الدمج. من بين جميع عمليات الدمج الممكنة، عند كل خطوة، تستهدف خوارزمية HUFFMAN عملية الدمج ذات التكلفة الأقل. يتبين التوطئة التالية أن لمسألة بناء أرمزة سبقة مثلى خاصية البنية الجزئية المثلى.

### توطئة 3.16

لتكن  $C$  أبجدية بتواتر  $c.freq$  لكل حرف  $c \in C$ . وليكن  $x$  و  $y$  حرفون من  $C$  بأقل التواترات. ولتكن الأبجدية  $C'$  التي خلُف منها  $x$  و  $y$ ، وأضيف إليها حرف جديد  $z$ ، أي:  $C' = C - \{x, y\} \cup \{z\}$ . نعرف  $freq$  للأبجدية  $C'$  كما هو معرف لـ  $C$  باستثناء أن  $z.freq = x.freq + y.freq$ . لتكن  $T'$  أية شجرة تمثل رمازاً سبقياً أمثل للأبجدية  $C'$ . عندها تمثل الشجرة  $T$ ، الناتجة من  $T'$  بالاستعاضة عن عقدة الورقة  $z$  بعقدة داخلية أبناؤها  $x$  و  $y$ ، رمازاً سبقياً أمثل للأبجدية  $C$ .

**البرهان** نبيّن أولاً أنه يمكن التعبير عن  $B(T)$  تكلفة الشجرة  $T$  بدلالة  $B(T')$  تكلفة الشجرة  $T'$  بالأخذ بالحسبان تكاليف مكونات المعادلة (4.16). لكل حرف  $c \in C - \{x, y\}$ ، لدينا  $d_T(c) = d_{T'}(c)$ ، ولذلك، فإن  $c.freq \cdot d_T(c) = c.freq \cdot d_{T'}(c)$ . ولما كان  $d_T(x) = d_T(y) = d_{T'}(z) + 1$ ، فيكون لدينا:

$$\begin{aligned} x.freq \cdot d_T(x) + y.freq \cdot d_T(y) &= (x.freq + y.freq)(d_{T'}(z) + 1) \\ &= z.freq \cdot d_{T'}(z) + (x.freq + y.freq), \end{aligned}$$

ومنه نستنتج أن:

$$B(T) = B(T') + x.freq + y.freq$$

أو العلاقة المكافئة:

$$B(T') = B(T) - x.freq - y.freq .$$

سنوهن التوظفة الآن بطريقة نقض الفرض. لنفترض أن  $T$  لا تُقَلِّل رمازًا سبقيًا أمثلًا للأبجدية  $C$ . إذن توجد شجرة مُثَلِّي  $T''$  بحيث  $B(T'') < B(T)$ . ومن دون إنقاص عمومية المسألة (باستخدام التوظفة 2.16)، نعتبر  $x$  و  $y$  أختين في الشجرة  $T''$ . لتكن  $T'''$  الشجرة  $T''$  بالاستعاضة عن الأب المشترك لـ  $x$  و  $y$  بالورقة  $z$  بتواتر  $z.freq = x.freq + y.freq$ . وعندها يكون:

$$\begin{aligned} B(T''') &= B(T'') - x.freq - y.freq \\ &< B(T) - x.freq - y.freq \\ &= B(T') , \end{aligned}$$

وهذا يناقض الفرض أن  $T'$  تقلل رمازًا سبقيًا أمثل لـ  $C$ . وبذلك، يجب أن تُقَلِّل  $T$  رمازًا سبقيًا أمثل للأبجدية  $C$ .

#### مبرهنة 4.16

تُنتِج الإجراءية HUFFMAN رمازًا سبقيًا أمثل.

البرهان يمكن إثبات هذه المبرهنة مباشرة من التوظفتين 2.16 و 3.16.

#### تعاريف

##### 1-3.16

ذكرنا في برهان التوظفة 2.16 أنه إذا كان  $x.freq = b.freq$  وُجِبَ أن يكون  $a.freq = b.freq = x.freq = y.freq$ . وضح السبب.

##### 2-3.16

برهن أنه لا يمكن أن توافق شجرة ثنائية غير ملأى رمازًا سبقيًا أمثل.

##### 3-3.16

ما هو رماز هوفمان الأمثل للمجموعة التالية من الترددات، التي تعتمد على أعداد فيبوناتشي Fibonacci الثمانية الأولى؟

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21

هل يمكنك تعميم إجابتك لإيجاد الرماز الأمثل حين تكون التواترات هي أعداد فيبوناتشي الـ  $n$  الأولى؟

##### 4-3.16

برهن أنه يمكن أيضًا حساب التكلفة الكلية لشجرة رماز، على أنها مجموع تواتري ابني كل عقدة داخلية.

### 5-3.16

برهن أنه إذا رتبنا محارف أبجدية بحيث تكون تواتراتها متناقصة باطراد، فتحة رماز أمثل تكون كلمات رمازه متزايدة الطول باطراد.

### 6-3.16

نفترض أن لدينا رمازًا سبقًا أمثل على مجموعة  $C = \{0, 1, \dots, n-1\}$  من المحارف، وأتأ نود إرسال هذا الرماز باستخدام أقل عدد ممكن من البتات. بين كيف يمكن تمثيل أي رماز سيق أمثل على  $C$  باستخدام  $2n-1 + n \lg n$  بتًا فقط. (تلميح: استخدم  $2n-1$  بتًا لتحديد بنية الشجرة، فيما يجري استكشافها بالمسير عرھا.)

### 7-3.16

عُسم خوارزمية هوفمان لكلمات رماز ثنائية (أي كلمات رماز تستخدم الرموز 0 و 1 و 2)، برهن أھا تعطي أرمزة ثنائية مثلى.

### 8-3.16

افترض أن لدينا ملف معطيات يتضمن متالية من محارف ذات ثمانية بتات بحيث تكون جميع المحارف الـ 256 تقريبًا بنفس الشيوع: أي التواتر الأعظم للمحارف أقل من ضعف التواتر الأدنى لها. برهن أن ترميز هوفمان في هذه الحالة ليس أكثر فعالية من الرماز العادي المحدد الطول  $\lceil \lg n \rceil$  بتات.

### 9-3.16

بين أنه لا يوجد أسلوب ضغط يتوقع أن يضغط ملف محارف ذات ثمانية بتات مختارة عشوائيًا ولا حتى ليقلص منه بتًا واحدًا. (تلميح: قارن عدد الملفات بعدد الملفات المرتزة الممكنة.)

## \* 4.16 الكيانات المصفوفية والطرائق الشرهة

نعرض في هذا المقطع نظرية جميلة عن الخوارزميات الشرهة. تصف هذه النظرية حالات كثيرة تعطي فيها الطريقة الشرهة حلولًا مثلى. وهي تتطلب بنى تراكيبية (توافقية) تسمى كيانات مصفوفية "matroids". ومع أن هذه النظرية لا تشمل جميع الحالات التي تُطبق فيها الطرائق الشرهة (فهي لا تشمل مثلاً مسألة اختيار النشاط في المقطع 1.16 أو مسألة ترميز هوفمان في المقطع 3.16)، إلا أھا تشمل الكثير من الحالات الهامة عمليًا. يضاف إلى ذلك، أن هذه النظرية توسعت لتشمل تطبيقات عديدة كثيرة؛ انظر للملاحظات في آخر هذا الفصل لمعرفة المراجع.

### الكيانات المصفوفية

الكيان المصفوفي Matroid هو زوج مرتب  $M = (S, I)$  يحقق الشروط التالية:

1.  $S$  مجموعة منتهية.

2.  $I$  جماعة غير خالية من المجموعات الجزئية من  $S$ ، تسمى **المجموعات الجزئية المستقلة** *independent subsets* من  $S$ . بحيث أنه إذا كان  $B \in I$  وكان  $A \subseteq B$ ، فإن  $A \in I$ . ونقول إن  $I$  وراثية *hereditary* إذا حققت هذه الخاصية. لاحظ أن المجموعة الخالية  $\emptyset$  هي بالضرورة عنصر في  $I$ .

3. إذا كان  $A \in I$  و  $B \in I$  و  $|A| < |B|$  فتمتد عنصر  $x \in B - A$  بحيث  $A \cup \{x\} \in I$ . نقول أن  $M$  تحقق خاصية التبادل *exchange property*.

يعود فضل ابتكار كلمة الكيان للمصفوف *"matroid"* إلى Hassler Whitney؛ فقد كان يدرس الكيان **المصفوفي المصفوفاتي** *matric matroid*، الذي تكون فيه عناصر  $S$  أسطر مصفوفة معطاة وتكون مجموعة الأسطر مستقلة إذا كانت مستقلة عطفياً بالمعنى المعتاد. هذه البنية تعرف كياناً مصفوفياً، يُطلب إليك في التمرين 2-4.16 بيان ذلك.

لغة مثال آخر على الكيانات للمصفوف هو **الكيان المصفوفي البياني** *graphic matroid*  $M_G = (S_G, I_G)$  المعروف بدلالة البيان غير الموجه  $G = (V, E)$  كما يلي:

- المجموعة  $S_G$  هي المجموعة  $E$ ، مجموعة الوصلات في  $G$ .
  - إذا كانت  $A$  مجموعة جزئية من  $E$ ، فإن  $A \in I_G$  إذا وفقط إذا كانت  $A$  خالية من الحلقات. أي إن مجموعة الوصلات في  $A$  مستقلة إذا وفقط إذا كان البيان الجزئي  $G_A = (V, A)$  يكون غابة.
- إن الكيان للمصفوف البياني  $M_G$  وثيق الصلة بمسألة شجرة المسح الصغرى، المشروحة بالتفصيل في الفصل 23.

### مبرهنة 5.16

إذا كان  $G = (V, E)$  بياناً غير موجه، فإن  $M_G = (S_G, I_G)$  كياناً مصفوفياً.

**البرهان** من الواضح أن  $S_G = E$  مجموعة منتهية، وأن  $I_G$  وراثية، لأن المجموعة الجزئية من غابة هي أيضاً غابة. وبطريقة أخرى، فإن حذف وصلات من مجموعة وصلات خالية من الحلقات لا يمكن أن ينشئ حلقات.

وهكذا فما علينا سوى أن نبين أن  $M_G$  تحقق خاصية التبادل. لنفرض أن  $G_A = (V, A)$  و  $G_B = (V, B)$  غابتان في  $G$  وأن  $|B| > |A|$ . أي إن  $A$  و  $B$  هما مجموعتا وصلات خالية من الحلقات، و  $B$  تحتوي على وصلات أكثر من  $A$ .

إن الغابة  $F = (V_F, E_F)$  تتضمن تماماً  $|V_F| - |E_F|$  شجرة. وليبان ذلك، نفرض أن  $F$  تتضمن  $t$  شجرة، حيث للشجرة  $t$  عقدة  $v_t$  و صلة. فيكون لدينا

$$\begin{aligned}
 |E_F| &= \sum_{i=1}^t e_i \\
 &= \sum_{i=1}^t (v_i - 1) \quad (\text{اعتماداً على المبرهنة (ب.2)}) \\
 &= \sum_{i=1}^t v_i - t \\
 &= |V_F| - t,
 \end{aligned}$$

وهذا يقتضي أن  $t = |V_F| - |E_F|$ . وبذلك فإن الغابة  $G_A$  تتضمن  $|V| - |A|$  شجرة، والغابة  $G_B$  تتضمن  $|V| - |B|$  شجرة.

ولما كانت الغابة  $G_B$  تتضمن أشجاراً أقل من الغابة  $G_A$ ، ونجب أن تتضمن الغابة  $G_B$  شجرة ما  $T$  عقدها في شجرتين مختلفتين من الغابة  $G_A$ . يضاف إلى ذلك أنه لما كانت  $T$  مترابطة، ونجب أن تتضمن وصلة  $(u, v)$  بحيث تكون العقدتان  $u$  و  $v$  في شجرتين مختلفتين من الغابة  $G_A$ . ولما كانت الوصلة  $(u, v)$  تصل عقدتين من شجرتين مختلفتين من الغابة  $G_A$ ، فيمكننا إضافة الوصلة  $(u, v)$  إلى الغابة  $G_A$  دون إنشاء حلقة. وبهذا، نحقق  $M_G$  خاصية التبادل، ويتم البرهان على أن  $M_G$  كيان مصفوفي. ■

إذا كان لدينا كيان مصفوفي  $(M, S, I)$ ، فإننا نسمي العنصر  $x \in A$  توسع  $extension$  المجموعة  $A \in I$  إذا أمكن إضافة  $x$  إلى  $A$  بحيث نحافظ على الاستقلال؛ أي إن  $x$  توسع لـ  $A$  إذا كان  $A \cup \{x\} \in I$ . كمثال على ذلك، لناخذ كياناً مصفوفياً بيانياً  $M_G$ ؛ فإذا كانت  $A$  مجموعة مستقلة من الوصلات، فإن الوصلة  $■$  توسع للمجموعة  $A$  إذا ونقط إذا لم تكن  $e$  من  $A$  وإذا لم تؤد إضافة  $e$  إلى  $A$  إلى إنشاء حلقة. إذا كانت  $A$  مجموعة جزئية مستقلة في كيان مصفوفي  $M$ ، فإننا نقول عن  $A$  إنها **مغطى**  $maximal$  إذا لم يكن لها أي توسع. أي تكون  $A$  مغطى إذا لم تكن محتواة في أية مجموعة جزئية مستقلة من  $M$  أكبر منها. الخاصية التالية مفيدة في أحيان عديدة.

#### مبرهنة 6.16

كل المجموعات الجزئية المستقلة المغطى من كيان مصفوفي لها الحجم نفسه.

**البرهان** نفترض العكس؛ أي إن  $A$  مجموعة جزئية مستقلة مغطى في  $M$ ، وتوجد مجموعة جزئية مستقلة مغطى أكبر منها  $B$  في  $M$ . وهذا يقتضي بموجب خاصية التبادل أنه يمكننا توسيع  $A$  إلى مجموعة مستقلة أكبر  $A \cup \{x\}$ ، حيث  $x \in B - A$ . وهذا يناقض افتراض أن  $A$  مغطى. ■

ك توضيح لهذه المبرهنة، لناخذ كياناً مصفوفياً بيانياً  $M_G$  لبيان مترابط غير موجه  $G$ . يجب أن تكون كل

بمجموعة جزئية مستقلة عُظُمَت في  $M_G$  شجرة حرة لما  $|V| - 1$  وصلةً تمامًا تصل جميع عقد  $G$ . نسمي مثل هذه الشجرة **شجرة مسح** *spanning tree* في  $G$ .

نقول إن الكيان المصفوي  $(S, I)$   $M = (S, I)$  **مُثَقِّل** *weighted* إذا كان مرفقًا بدالة ثقل (وزن)  $w$  تسند ثقلًا موجبًا تمامًا  $w(x)$  لكل عنصر  $x \in S$ . تتوسع دالة الثقل  $w$  إلى المجموعات الجزئية في  $S$  بالجمع

$$w(A) = \sum_{x \in A} w(x)$$

لأية مجموعة جزئية  $A \subseteq S$ . على سبيل المثال، إذا جعلنا  $w(e)$  ترمز إلى ثقل الوصلة  $e$  في الكيان المصفوي البياني  $M_G$ ، فإن  $w(A)$  هو الثقل الكلي للوصلات في مجموعة الوصلات  $A$ .

### الخوارزميات الشجرة على كيان مصفوفي مُثَقِّل

يمكن صياغة العديد من المسائل -التي يعطي النهج الشجرة حلولاً مُثَقِّلًا لها- على أنها مسائل إيجاد مجموعة جزئية مستقلة بقل أعظم في كيان مصفوفي مُثَقِّل. أي إنه يوجد كيان مصفوفي مُثَقِّل  $M = (S, I)$ ، ونود إيجاد مجموعة مستقلة  $A \in I$  بحيث يكون  $w(A)$  ذا قيمة عُظُمَت. نسمي مثل هذه المجموعة الجزئية، التي هي مستقلة ولها ثقل أعظم ممكن، مجموعة **مُثَقِّل** *optimal* للكيان المصفوفي. ولما كان ثقل أي عنصر  $x \in S$   $w(x)$  موجبًا، فإن أية مجموعة جزئية مُثَقِّل هي دائمًا مجموعة جزئية مستقلة عُظُمَت - من المفيد دومًا جعل  $A$  كبيرة قدر الإمكان.

على سبيل المثال، في مسألة **شجرة المسح الصغرى** *minimum-spanning-tree problem*، لدينا بيان مترابط غير موجه  $G = (V, E)$ ، ودالة طول  $w$  بحيث تكون  $w(e)$  الطول (الموجب) للوصلة  $e$ . (نستخدم المصطلح "طول" هنا لنشير إلى أنقال الوصلة الأصلية في البيان، ونحتفظ بالمصطلح "ثقل" للإشارة إلى الأنقال في الكيان المصفوفي المرافق.) ونرغب في إيجاد مجموعة جزئية من الوصلات التي تصل كل العقد معًا وبحيث يكون لها طول كلي أصغر. ولعرض هذه المسألة على أنها مسألة إيجاد مجموعة جزئية مُثَقِّل في كيان مصفوفي، لناخذ الكيان المصفوفي المثلل  $M_G$  مع دالة الثقل  $w$ ، حيث  $w_0 - w(e) = w'(e)$  و  $w_0$  أكبر من الطول الأعظم لأي وصلة. في هذا الكيان المصفوفي المثلل، جميع الأنقال موجبة، والمجموعة الجزئية المثلل هي شجرة مسح بطول كلي أصغر في البيان الأصلي. وتعبير أدق، تُقابل كل مجموعة جزئية مستقلة عُظُمَت  $A$  شجرة مسح، لما  $|V| - 1$  وصلة، ولما كان:

$$\begin{aligned} w'(A) &= \sum_{e \in A} w'(e) \\ &= \sum_{e \in A} (w_0 - w(e)) \end{aligned}$$

$$\begin{aligned} &= (|V| - 1)w_0 - \sum_{e \in A} w(e) \\ &= (|V| - 1)w_0 - w(A) \end{aligned}$$

في حالة أية مجموعة جزئية مستقلة عظمى  $A$ ، فإن المجموعة الجزئية المستقلة التي تجعل الكمية  $w'(A)$  عظمى، عليها أن تجعل  $w(A)$  صغرى. وبذلك، فإن أي خوارزمية توجد مجموعة جزئية مثلى  $A$  في كيان مصفوفي اعتباطي يمكنها أن تحل مسألة للمسح الصغرى.

يعرض الفصل 23 خوارزميات لمسألة شجرة المسح الصغرى، ولكننا نعرض هنا خوارزمية شرهة، تصلح لأي كيان مصفوفي مثقل. تأخذ الخوارزمية كياناً مصفوفياً  $M = (S, I)$  على أنه دخل لها، مع دالة ثقل موجب مرافق  $w$ ، وتعيد مجموعة جزئية مثلى  $A$ . نرسم في شبه رمازنا إلى مكونات  $M.S$  و  $M.I$  وللدالة الثقل بـ  $w$ . إن هذه الخوارزمية شرهة، لأنها تأخذ كل عنصر  $x \in S$  بدوره في الترتيب للتناقص باطراد للوزن، وتضيف مباشرة إلى المجموعة  $A$  التي هي في قيد البناء إذا كانت المجموعة  $A \cup \{x\}$  مستقلة.

**GREEDY( $M, w$ )**

- 1  $A = \emptyset$
- 2 sort  $M.S$  into monotonically decreasing order by weight  $w$
- 3 for each  $x \in M.S$ , taken in monotonically decreasing order by weight  $w(x)$
- 4     if  $A \cup \{x\} \in M.I$
- 5          $A = A \cup \{x\}$
- return  $A$

يتحصن السطر 4 ما يلي: هل نحافظ إضافة أي عنصر  $x$  إلى المجموعة  $A$  على استقلالية  $A$ ؟ فإذا بقيت  $A$  مستقلة، فإن السطر 5 يضيف  $x$  إلى  $A$ ، وإلاً نعمل  $x$ . ولما كانت المجموعة الحالية مستقلة، وكان كل تكرار في الحلقة for يحافظ على استقلالية  $A$ ، فإن المجموعة الجزئية مستقلة دوماً (بالاستقراء). ولذلك، تعيد الخوارزمية GREEDY دائماً مجموعة مستقلة  $A$ . وسنجد بعد قليل أن  $A$  مجموعة جزئية بثلث أعظم ممكن، وبذلك تكون  $A$  مجموعة جزئية مثلى.

من السهل تحليل زمن تنفيذ الخوارزمية GREEDY. لنرمز إلى  $|S|$  بـ  $n$ . تستغرق مرحلة الفرز في الخوارزمية GREEDY زمناً  $O(n \lg n)$ . يُنفَّذ السطر 4، مرة، مرة لكل عنصر من  $S$ . يتطلب كل تنفيذ للسطر 4 فحصاً ما يلي: هل المجموعة  $A \cup \{x\}$  مستقلة أم لا؟ إذا استغرقت كل عملية فحص زمناً  $O(f(n))$ ، فإن كامل الخوارزمية تنفذ بزم  $O(n \lg n + n f(n))$ . نبرهن الآن أن الخوارزمية GREEDY تعيد مجموعة جزئية مثلى.

**توطئة 7.16 (الكيانات المصفوفية تتمتع بخاصية الخيار الشره)**

لنفترض  $M = (S, I)$  كياناً مصفوفياً مثقلاً، بدالة ثقل  $w$ ، وأن  $S$  مفروزة بحسب الترتيب للتناقص باطراد



للتقل. ليكن  $x$  أول عنصر في  $S$  بحيث تكون المجموعة  $\{x\}$  مستقلة، إن وُجد. فإذا وُجد  $x$ ، فتوجد مجموعة جزئية مثلى  $A$  من  $S$  تتضمن  $x$ .

**البرهان** إذا لم يكن مثل هذا العنصر  $x$  موجوداً، فإن المجموعة الجزئية للمستقلة الوحيدة تكون هي المجموعة الخالية، وتكون التغطية صحيحة بداهة. وإلا، فلتكن  $B$  مجموعة جزئية مثلى غير خالية. ونفترض أن  $x \notin B$ ؛ وإلا فيحصل  $A = B$  نحصل على مجموعة جزئية مثلى من  $S$  تتضمن  $x$ .

لا يوجد عنصر من  $B$  وزنه أكبر من  $w(x)$ . وليكن ذلك، نلاحظ أن  $y \in B$  يقتضي أن تكون  $\{y\}$  مستقلة، لأن  $B \in I$  و  $I$  وراثية. وعلى ذلك، فإن خيارنا ل  $x$  يتضمن أن يكون  $w(x) \geq w(y)$  لأي عنصر  $y \in B$ .

نشئ المجموعة  $A$  كما يلي: نبدأ بـ  $A = \{x\}$ . تبعاً لطريقة اختيار  $x$ ، تكون  $A$  مستقلة. وباستخدام خاصية التبادل، نوجد تكرارياً عنصراً جديداً من  $\square$  يمكن إضافته إلى  $A$  إلى أن يصبح  $|A| = |B|$ ، مع الاحتفاظ باستقلالية  $A$ . عند هذه النقطة تكون  $A$  هي نفس  $B$  باستثناء أن  $A$  تتضمن  $x$ ، و  $B$  تتضمن عنصراً آخر  $y$ ، أي إن  $A = \square - \{y\} \cup \{x\}$  لعنصر ما  $y \in \square$ ، وبذلك يكون:

$$\begin{aligned} w(A) &= w(B) - w(y) + w(x) \\ &\geq w(B). \end{aligned}$$

ولما كانت المجموعة  $\square$  مثلى، ونحِب أن تكون المجموعة  $A -$  التي تتضمن  $x -$  مثلى أيضاً. ■

منهين لاحقاً أنه إذا لم يكن عنصر ما خياراً في البداية، فلن يكون خياراً لاحقاً.

### توطئة 8.16

ليكن  $M = (S, I)$  أي كيان مصفوي. إذا كان  $x$  عنصراً من  $S$  وتوسفاً لمجموعة جزئية مستقلة  $A$  من  $S$ ، فإن  $\square$  توسع أيضاً للمجموعة الخالية  $\emptyset$ .

**البرهان** لما كان  $x$  توسفاً ل  $A$ ، فإن  $A \cup \{x\}$  مجموعة مستقلة. ولما كانت  $I$  وراثية، ونحِب أن تكون  $\{x\}$  مستقلة، وبذلك فإن  $\{x\}$  توسع للمجموعة الخالية  $\emptyset$ . ■

### نتيجة 9.16

ليكن  $M = (S, I)$  أي كيان مصفوي. إذا كان  $x$  عنصراً من  $S$  بحيث لا يكون  $x$  توسفاً للمجموعة الخالية  $\emptyset$ ، فإن  $x$  ليس توسفاً لأي مجموعة جزئية مستقلة  $A$  من  $S$ .

**البرهان** هذه النتيجة هي ببساطة الاقتضاء للمعكس الموجب للـ contrapositive للتوطئة 8.16. ■

تعني النتيجة 9.16 أن أي عنصر إذا لم يكن بالإمكان استخدامه فوراً، فلن يُستخدم أبداً. لذلك، لا يمكن أن تخطئ خوارزمية GREEDY بترك أي عناصر بدئية في  $S$  ليست توسعاً لـ  $B$  جانباً، لأنها لن تُستخدم أبداً.

#### توطئة 10.16 (الكيانات المصفوفية تتمتع بخاصية البنية الجزئية المثلى)

ليكن  $x$  أول عنصر في  $S$  تختاره الخوارزمية GREEDY للكيان المصفوفي للثقل  $M = (S, I)$ . المسألة المتبقية وهي إيجاد مجموعة جزئية مستقلة ذات وزن أعظم تتضمن  $x$ ، تُختصر إلى مسألة إيجاد مجموعة جزئية مستقلة ذات وزن أعظم من الكيان المصفوفي للثقل  $M' = (S', I')$ ، حيث

$$S' = \{y \in S : \{x, y\} \in I\},$$

$$I' = \{B \subseteq S - \{x\} : B \cup \{x\} \in I\}.$$

ودالة الثقل لـ  $M'$  هي دالة الثقل لـ  $M$ ، مقصورةً على  $S'$ . (نسمي  $M'$  تقليص  $M$  contraction  $M$  اعتماداً على العنصر  $x$ .)

**البرهان** إذا كانت  $A$  أية مجموعة جزئية مستقلة ذات وزن أعظم من  $M$  تتضمن  $x$ ، عندها تكون  $A' = A - \{x\}$  مجموعة جزئية مستقلة في  $M'$ . وبالعكس أية مجموعة جزئية مستقلة  $A'$  من  $M'$  تعطي مجموعة جزئية مستقلة  $A = A' \cup \{x\}$  من  $M$ . ولما كان لدينا في كلتا الحالتين  $w(A) = w(A') + w(x)$ ، فإن الحل ذا الوزن الأعظم في  $M$  الذي يتضمن  $x$  يؤدي إلى حل ذي ثقل أعظم في  $M'$  وبالعكس. ■

#### مبرهنة 11.16 (صحة الخوارزمية الشرة على الكيانات المصفوفية)

إذا كان  $M = (S, I)$  كياناً مصفوفياً مثقلاً بدالة ثقل  $w$ ، فإن الإجراء  $\text{GREEDY}(M, w)$  يعيد مجموعة جزئية مثلى.

**البرهان** نستنتج من النتيجة 9.16 أن كل العناصر التي يُهملها GREEDY منذ البداية (لأنها ليست توسعاً لـ  $B$ ) يمكن تجاهلها، لأنها لن تكون مفيدة. وحين يختار GREEDY العنصر الأول  $x$ ، فإن التوطئة 7.16 تقتضي أن الخوارزمية لا تخطئ بإضافة  $x$  إلى  $A$ ، لوجود مجموعة جزئية مثلى تتضمن  $x$ . أمراً، ينتج عن التوطئة 10.16 أن المسألة المتبقية هي مسألة إيجاد مجموعة جزئية مثلى في الكيان المصفوفي  $M'$ ، وهو تقليص  $M$  اعتماداً على  $x$ . وبعد أن يُعطى الإجراء GREEDY المجموعة  $A$  القيمة  $\{x\}$ ، فإنه يمكننا تفسير جميع الخطوات المتبقية على أنها تعمل في الكيان المصفوفي  $M' = (S', I')$ ، لأن المجموعة  $B$  تكون مستقلة في  $M'$  إذا وفقط إذا كانت  $B \cup \{x\}$  مستقلة في  $M$ ، لكل المجموعات  $I' \subseteq B$ . وبذلك، ستجد العملية التالية من GREEDY مجموعة جزئية مستقلة ذات ثقل أعظم على  $M'$ ، وستنتج عن بحمل الإجراء Greedy مجموعة جزئية مستقلة مثلى في  $M$ . ■

## تعاريف

## 1-4.16

يُبين أن  $(S, I_k)$  هو كيان مصفوفي، حيث  $S$  مجموعة منتهية و  $I_k$  مجموعة كل المجموعات الجزئية من  $S$  التي حجمها  $k$  على الأكثر، حيث  $k \leq |S|$ .

## \* 2-4.16

لكن لدينا للمصفوفة  $T$  ذات البعدين  $m \times n$ ، للتعرف على حقل ما (مثل حقل الأعداد الحقيقية). يُبين أن  $(S, I)$  هو كيان مصفوفي، حيث  $S$  مجموعة الأعمدة في  $T$  و  $A \in I$  إذا وفقط إذا كانت أعمدة  $A$  مستقلة خطيًا.

## \* 3-4.16

يُبين أنه إذا كان  $(S, I)$  كيانًا مصفوفيًا، فإن  $(S, I')$  كيان مصفوفي، حيث:

$$I' = \{A' : S - A' \text{ contains maximal } A \in I\}.$$

أي إن المجموعات المستقلة المُغطى في  $(S, I')$  هي متممات المجموعات للمستقلة المُغطى في  $(S, I)$ .

## \* 4-4.16

لكن  $S$  مجموعة منتهية، ولتكن  $S_1, S_2, \dots, S_k$  تجزئة  $S$  إلى مجموعات جزئية منفصلة غير خالية. عرّف البنية  $(S, I)$  بشرط  $I = \{A : |A \cap S_i| \leq 1 \text{ for } i = 1, 2, \dots, k\}$ . يُبين أن  $(S, I)$  كيان مصفوفي؛ أي إن مجموعة كل المجموعات  $A$  التي تتضمن عضوًا (عنصرًا) واحدًا على الأكثر في كل كتلة من التجزئة، تُحدّد المجموعات المستقلة للكيان المصفوفي.

## \* 5-4.16

يُبين كيف تتحول دالة النقل لمسألة كيان مصفوفي مثقل، حيث الحل الأمثل المرغوب هو مجموعة جزئية مستقلة مُغطى ذات نقل أصغر، لجعله مسألة كيان مصفوفي مثقل معياري. ناقش بعناية صحة تحويلك.

## \* 5.16 مسألة جدولة المهام

من المسائل المهمة التي يمكن حلها باستخدام الكيانات للمصفوفية مسألة جدولة مهام في واحدة الزمن، جدولة مُثَقِّل، باستخدام معالج واحد، حيث لكل مهمة حدّ انتهاء مُحدد، يجب بعده دفع غرامة إذا لم تُنفذ المهمة قبل حدّ انتهائها. تبدو المسألة معقدة، إلا أنه يمكننا حلها بطريقة غاية في البساطة وذلك بتحويلها إلى كيان مصفوفي واستخدام خوارزمية شرية.

**المهمة في واحدة الزمن unit-time task** هي عمل، مثل برنامج، يجب تنفيذه على حاسوب ويتطلب

تماماً واحدة زمن لاستكمالها. إذا كانت لدينا مجموعة منتهية  $S$  من المهام في واحدة الزمن، فإن *جدولة schedule* المجموعة  $S$  هي تبديل على  $S$  يحدد الترتيب الذي يجب إنجاز هذه المهام وفقه. تبدأ المهمة الأولى في الجدولة في اللحظة 0 وتنتهي في اللحظة 1، وتبدأ المهمة الثانية في اللحظة 1 وتنتهي في اللحظة 2، وهكذا...

لمسألة *جدولة المهام في واحدة الزمن، بحدود انتهاء وغرامات، على معالج واحد scheduling* *unit time tasks with deadlines and penalties for a single processor*، المدخلات التالية:

- مجموعة  $S = \{a_1, a_2, \dots, a_n\}$  من  $n$  مهمة في واحدة الزمن.
  - مجموعة من  $n$  عدداً طبيعياً هي *حدود انتهاء deadlines*  $d_1, d_2, \dots, d_n$  بحيث تحقق كل  $d_i$ :  $1 \leq d_i \leq n$ ، ويُفترض أن تنتهي المهمة  $a_i$  قبل اللحظة  $d_i$  و
  - مجموعة من  $n$  نقلاً غير سالب، أو *غرامة penalty*  $w_1, w_2, \dots, w_n$  بحيث نتأكد الغرامة  $w_i$  إذا لم تنته المهمة  $a_i$  بحلول  $d_i$ ، ولا تتعرض لأية غرامة إذا انتهت المهمة قبل حلول حدّ انتهائها.
- نرغب في إيجاد جدول  $J$  يصنّف الغرامة الكلية الناتجة عن تجاوز حدود الانتهاء.

لنأخذ جدولاً معيناً. نقول عن مهمة إنها *متأخرة late* في هذا الجدول إذا انتهت بعد حدّ انتهائها، وإلا فنقول إن هذه المهمة *مبكرة early* في الجدول. يمكن دائماً وضع أي جدول اعتباطي بالشكل *المبكر أولاً early first form*. وفي هذا الشكل نسبق للمهام المبكرة المهام المتأخرة. ولبيان ذلك، لاحظ أنه إذا لحقت مهمة مبكرة  $a_i$  مهمة متأخرة  $a_j$  عندها يمكننا للمبادلة بين  $a_i$  و  $a_j$ ، وبقي  $a_i$  مبكرة و  $a_j$  متأخرة.

يضاف إلى ذلك، أنه يمكن دوماً تحويل أي جدول اعتباطي إلى *الشكل القانوني canonical form*، الذي تسبق فيها المهام المبكرة المهام المتأخرة، وتُجدول المهام المبكرة بالترتيب المتزايد باطراد لحدود انتهائها. وللقيام بذلك، نضع الجدول بالشكل المبكر أولاً. ثم إذا وجدنا مهمتين مبكرتين  $a_i$  و  $a_j$  تنتهيان في الجدول في اللحظات  $k$  و  $k+1$  على الترتيب بحيث يكون  $d_i < d_j$ ، فإننا نبدل مواضع  $a_i$  و  $a_j$ . ولما كانت  $a_j$  مبكرة قبل الإبدال، فإن  $d_j \leq k+1$ . إذن،  $k+1 < d_i$ ، وبذلك تبقى  $a_i$  مبكرة بعد الإبدال. وبسبب تحريك المهمة  $a_j$  إلى موضع أبكر في الجدول، فإنها تبقى مبكرة بعد الإبدال.

وهكذا يتّوّل البحث عن الجدول الأمثل إلى إيجاد مجموعة مهام  $A$  التي يجب أن تكون مبكرة في الجدول الأمثل. فإذا حدّدنا  $A$ ، أمكننا إنشاء الجدول الفعلي بسرد عناصر  $A$  بالترتيب المتزايد باطراد لحدود الانتهاء، ثم بسرد للمهام المتأخرة (أي  $S - A$ ) بأي ترتيب، وهذا يُنتج ترتيباً قانونياً للجدول الأمثل.

نقول عن مجموعة من المهام  $A$  إنها *مستقلة independent* إذا وُجد جدول لهذه المهام لا يكون فيها أية مهمة متأخرة. من الواضح أن مجموعة المهام المبكرة في أي جدول تكون مجموعة مستقلة من المهام. ليرمز بـ  $I$  إلى مجموعة كل المجموعات المستقلة من المهام.

ندرس فيما يلي مسألة الحكم على مجموعة معطاة من المهام  $A$  بأنها مستقلة. نرسم  $N_t(A)$  إلى عدد المهام في  $A$  التي لا يتجاوز حد انتهائها  $t$  حيث  $t = 0, 1, \dots, n$ . لاحظ أن  $N_0(A) = 0$  مهما كانت المجموعة  $A$ .

### 12.16 توطئة

مهما كانت مجموعة المهام  $A$ ، فإن العبارات التالية متكافئة.

1. المجموعة  $A$  مستقلة.
2.  $N_t(A) \leq t$  حيث  $t = 0, 1, \dots, n$ .
3. إذا جداولت المهام في  $A$  بالترتيب المتزايد باطراد لحدود الانتهاء، فلن توجد أية مهمة متأخرة.

**البرهان** سنثبت أن (1) تقتضي (2) بطريقة الاقتضاء المعاكس الموجب. إذا كان  $N_t(A) > t$  للحظة ما  $t$ ، فلا توجد طريقة لبناء جدول لا يتضمن أي مهام متأخرة في المجموعة  $A$ ، بسبب وجود أكثر من  $t$  مهمة يجب أن تنتهي قبل  $t$ . لذلك فإن (1) تقتضي (2). فإذا كانت (2) صحيحة، ونحب أن تكون (3) صحيحة بالضرورة؛ إذ لا مجال للوقوع في "مأزق" عند جدولة المهام بالترتيب المتزايد باطراد لحدود الانتهاء، لأن (2) تقتضي أن حد النهاية ذا الترتيب  $t$  من حيث كبر القيمة يساوي  $t$  على الأكثر. أخيراً، (3) تقتضي بداهة (1). ■

باستخدام الخاصية 2 من التوطئة 12.16، يمكننا بسهولة حساب كون مجموعة مهام معطاة  $A$  مستقلة أم لا (انظر التمرين 2-5.16).

إن مسألة تصغير مجموع الفرامات عن المهام المتأخرة هي نفسها مسألة تعظيم maximizing مجموع الفرامات عن المهام المبكرة. ولهذا فإن البرهنة التالية تؤكد أنه يمكننا استخدام الخوارزميات الشرية لإيجاد مجموعة  $A$  من المهام المستقلة بحيث تكون الفرامة الكلية عظمى.

### 13.16 مبرهنة

إذا كانت  $S$  مجموعة مهام في واحدة الزمن مع حدود انتهاء، وكانت  $I$  مجموعة كل مجموعات المهام المستقلة، فإن النظام الموافق  $(S, I)$  هو كياناً مصفوفياً.

**البرهان** إن كل مجموعة جزئية من مجموعة مهام مستقلة هي مستقلة بالتأكيد. لبرهان خاصية التبادل، نفترض أن  $A$  و  $B$  مجموعتان مستقلتان من المهام، وأن  $|B| > |A|$ . ولتكن  $k$  أكبر  $t$  بحيث يكون  $N_t(B) \leq N_t(A)$ ، (إن هذه القيمة موجودة، لأن  $N_0(A) = N_0(B)$ ). ولما كان  $N_n(B) = |B|$  ولما كان  $N_n(A) = |A|$  و  $|B| > |A|$ ، فيجب أن يكون  $k < n$  و  $N_k(B) > N_k(A)$  لكل قيم  $j$  في

مهمة

7	6	5	4	3	2	1	$a_i$
6	4	1	3	4	2	4	$d_i$
10	20	30	40	50	60	70	$w_i$

الشكل 7.16 منتخ (مثال) عن مسألة جدولة مهام في واحدة الزمن، مع حدود انتهاء وغرامات لمعالج واحد.

المجال  $1 \leq j \leq n$ . لذا، فإن  $B$  تتضمن مهامًا محدود انتهاء نسوي  $1 + \square$  أكثر مما تتضمنه  $A$ . لشكل  $a_i$  مهمة في  $B - A$  مع حد انتهاء  $k + 1$ ، ولتكن  $A' = A \cup \{a_i\}$ .

نبيّن الآن أن  $A'$  يجب أن تكون مستقلة باستخدام الخاصية 2 من التوطئة 12.16. لدينا  $N_i(A') = N_i(A) \leq t$ ، ولأن  $0 \leq t \leq k$ ، مستقلة. ولدينا  $N_i(B) \leq N_i(A') \leq N_i(B)$ ، ولأن  $k < t \leq n$ ، فإن  $A'$  مستقلة، وهذا يتضمّن برهاننا أن  $(S, I)$  كيان مصغري. ■

يمكننا اعتمادًا على المبرهنة 11.16 استخدام خوارزمية شرهة، لإيجاد مجموعة مهام مستقلة ذات وزن أعظم. يمكننا بعدها إنشاء جدول أمثل تكون مهام  $A$  هي مهامه للبكرة. نُقد هذه الطريقة خوارزمية فعالة لجدولة مهام في واحدة الزمن مع حدود انتهاء وغرامات لمعالج واحد. إن زمن التنفيذ هو  $O(n^2)$  باستخدام GREEDY، لأن كل فحص من فحوص الاستقلالية الـ  $O(n)$  التي تقوم بها الخوارزمية يستغرق زمنًا  $O(n)$  (انظر التمرين 2-5.16). ثمة خوارزمية أسرع إنجازًا نجدها في للسألة 4-16.

يبين الشكل 7.16 مثالاً على مسألة جدولة مهام في واحدة الزمن، مع حدود انتهاء وغرامات لمعالج واحد. في هذا المثال، تختار الخوارزمية الشرهة للمهام  $a_1$  و  $a_2$  و  $a_3$  و  $a_4$ ، على الترتيب، ثم ترفض كلاً من  $a_5$  (لأن  $N_4(\{a_1, a_2, a_3, a_4, a_5\}) = 5$ ) و  $a_6$  (لأن  $N_4(\{a_1, a_2, a_3, a_4, a_6\}) = 5$ )، وأخيراً تقبل  $a_7$ . ويكون الجدول الأمثل النهائي هو:

$$(a_2, a_4, a_1, a_3, a_7, a_5, a_6),$$

والغرامة الكلية هي  $w_5 + w_6 = 50$ .

تمارين

1-5.16

حلّ مثال مسألة الجدولة للمعطة في الشكل 7.16 ولكن بإبدال الغرامة  $w_i$  بـ  $w_i - 80$ .

2-5.16

بيّن كيف تُستخدم الخاصية 2 من التوطئة 12.16 لتحديد كون مجموعة من المهام  $A$  مستقلة أم لا في زمن  $O(|A|)$ .

## مسائل

## 1-16 تبديل العملات

ادرس مسألة صرف  $n$  سنًا باستخدام أقل عدد من القطع النقدية. افترض أن قيمة كل قطعة نقدية هي عدد صحيح.

أ. وُصِفَ خوارزمية شرهة للصرف مكوّنة من: أرباع الدولار، وأعشاره dimes، ونيكلاته nickels (الثُّكَلَة تساوي خمسة سنتات)، وسنتاته pennies (السنت يساوي 1/100 من الدولار). برهن أن خوارزمتك تحقق حلاً أمثل.

ب. افترض أن كسور مقامات قطع النقود المتاحة (الفتات النقدية) هي قوى صحيحة لـ  $c$ ، أي إن المقامات هي  $c^0, c^1, \dots, c^k$  حيث  $c > 1$  و  $k \geq 1$  عدنان صحيحان. بَيِّنْ أن الخوارزمية الشرهة تحقق دائماً حلاً أمثل.

ت. أعطِ مجموعة من الفتات النقدية تجعل الخوارزمية الشرهة لا تعطي حلاً أمثل. يجب أن تتضمن مجموعتك سنًا بحيث يوجد حلٌّ لكل قيمة  $n$ .

ث. أعطِ خوارزمية بزمن  $O(nk)$  تحقّق عملية الصرف لأية مجموعة مؤلّفة من  $k$  قطعة نقدية مختلفة، بافتراض أن السنت أحد هذه القطع.

## 2-16 جدولة تصغير متوسط لحظات الإنهاء

لنفترض أن لديك مجموعة من المهام  $S = \{a_1, a_2, \dots, a_n\}$ ، حيث تتطلب المهمة  $a_i$ ، حين تبدأ، عددًا  $p_i$  من وحدات المعالجة لإنهائها. ولديك حاسوب واحد لتنفيذ هذه المهام عليه. يستطيع هذا الحاسوب تنفيذ مهمة واحدة في اللحظة معينة. ولتكن  $c_i$  لحظة إنهاء completion time المهمة  $a_i$ ، أي اللحظة التي تنتهي عندها معالجة المهمة  $a_i$ . والمطلوب هو تصغير متوسط زمن الإنهاء، أي تصغير  $\sum_{i=1}^n c_i$ . افترض، على سبيل المثال، أن لديك مهمتين  $a_1$  و  $a_2$ ، وأن  $p_1 = 3$  و  $p_2 = 5$ ، ولناخذ الجدول الذي تُنفَّذ فيه  $a_2$  أولاً تبعها  $a_1$ . عندها يكون  $c_2 = 5$  و  $c_1 = 8$ ، ومتوسط لحظات الإنهاء  $6.5 = (5 + 8)/2$ . وإذا نُفِّذت المهمة  $a_1$  أولاً، فإن  $c_1 = 3$  و  $c_2 = 8$ ، ومتوسط لحظات الإنهاء  $5.5 = (3 + 8)/2$ .

أ. أعطِ خوارزمية جدول المهام بحيث تصغر متوسط لحظات الإنهاء. يجب أن تُنفَّذ كلُّ مهمة دون استحواذ، أي حين تبدأ المهمة  $a_i$  يجب أن تُنفَّذ باستمرار خلال  $p_i$  وحدة زمن. أثبت أن خوارزمتك تصغر متوسط لحظات الإنهاء، وأعطِ زمن تنفيذ خوارزمتك.

ب. افترض الآن أن المهام ليست جميعها متاحة معاً. أي إن كلَّ مهمة لا يمكن أن تبدأ ما لم تُجَرَّ لحظة

**إصدارها  $r_i$  release time.** افترض أيضًا أننا نسمح بالاستحواف *preemption* بحيث يمكن تعليق مهمة ثم إعادة تشغيلها في وقت لاحق. فمثلًا، يمكن لمهمة  $a_i$  (زمن معالجتها  $p_i = 6$ ) وزمن إصدارها ( $r_i = 1$ ) أن تبدأ بالتنفيذ في اللحظة 1 ثم يُستحوذ عليها في اللحظة 4. يمكن استئنافها في اللحظة 10 والاستحواف عليها في اللحظة 11، وأخيرًا استئنافها في اللحظة 13 وإتمامها في اللحظة 15. جرى تنفيذ المهمة  $a_i$  خلال زمن كلي يساوي 6 وحدات زمن، إلا أن زمن تنفيذها قد قُسم إلى ثلاث قطع. نقول إن لحظة إنهاء  $a_i$  هو 15. أعطِ خوارزميةً تحسّل المهام بحيث تصغر متوسط لحظات الإنهاء في هذا السيناريو. برهن أن خوارزميةك تصغر متوسط لحظات الإنهاء، وأعطِ زمن تنفيذها.

### 3-16 البيانات الجزئية الخالية من الحلقات

أ. **مصفوفة الورود *incidence matrix*** لبيان غير موجه  $G = (V, E)$  هي مصفوفة  $M \in \mathbb{R}^{|V| \times |E|}$  حيث  $M_{ve} = 1$  إذا كانت الوصلة  $e$  واردة على العقدة  $v$ ، وإلا فإن  $M_{ve} = 0$ . ناقش ما يلي: تكون مجموعة من أعمدة  $M$  مستقلة خطيًا على حقل الأعداد الصحيحة بالمقاس 2، إذا وفقط إذا كانت مجموعة الوصلات الموافقة خالية من الحلقات.

ب. لنفترض أننا نرفق ثقلًا غير سالب  $w(e)$  بكل وصلةٍ من بيانٍ غير موجه  $G = (V, E)$ . أعطِ خوارزميةً فعالة لإيجاد مجموعة جزئية من  $E$  خالية من الحلقات وبحيث يكون ثقلها الكلي أعظمًا.

ت. ليكن  $G = (V, E)$  بيانًا موجهًا، وليكن  $(E, I)$  معرفًا بحيث يكون  $A \in I$  إذا وفقط إذا لم يتضمن  $A$  أية حلقات موجهة. أعطِ مثالًا على بيانٍ موجه  $G$  بحيث لا يكون النظام الموافق  $(E, I)$  كيانًا مصفوفيًا. حدّد الشرط غير الخلق من تعريف الكيان للمصفوف.

ث. إذا كانت **مصفوفة الورود *incidence matrix*** لبيانٍ موجه  $G = (V, E)$  من دون حلقات ذاتية هي مصفوفة  $|V| \times |E|$  بحيث يكون  $M_{ve} = -1$  إذا كانت الوصلة  $e$  تغادر العقدة  $v$  و  $M_{ve} = 1$  إذا كانت الوصلة  $e$  تدخل العقدة  $v$ ، و  $M_{ve} = 0$  فيما عدا ذلك. ناقش ما يلي: إذا كانت مجموعة جزئية من أعمدة  $M$  مستقلة خطيًا، فإن مجموعة الوصلات الموافقة لا تتضمن حلقةً موجهة.

ج. يقرّر التمرين 4.16-2 أن مجموعة مجموعات الأعمدة المستقلة خطيًا لأية مصفوفة  $M$  تكون كيانًا مصفوفيًا. فسر بعناية لماذا لا تتناقض نتائج الجزأين (ت) و (ث) فيما بينها. كيف يمكن ألا يتحقق التوافق الكامل بين مفهوم الوصلات الخالية من الحلقات ومفهوم الاستقلال الخطي لمجموعة الأعمدة الموافقة من مصفوفة الورود.

### 4-16 أنماط أخرى من مسألة الجدولة

لندرس الخوارزمية التالية للمسألة المذكورة في المقطع 5.16، وهي خاصة بجدولة مهام في واحدة الزمن مع حدود



اتهاء وغرامات. لتكن جميع الشرائع الزمنية (عددها  $n$ ) فارغة في البداية، حيث الشريعة الزمنية  $i$  هي الشريعة التي طولها واحدة الزمن وتنتهي في اللحظة  $i$ . نأخذ المهام مرتبةً بحسب الغرامات المتناقصة باطراد. ندرس المهمة  $a_i$  كما يلي: إذا وُجد للمهمة  $a_i$  شريعة زمنية عند حدّ انتهاء  $d_i$  (أو قبله) وما تزال فارغة، فإننا نُسند  $a_i$  إلى آخر شريعة تحقق ذلك، فنصبح الشريعة مملأة (أو مشغولة). وإذا لم نجد هذه الشريعة الزمنية، فإننا نُسند  $a_i$  إلى آخر شريعة لم تُملأ بعد.

أ. أثبت أن هذه الخوارزمية تعطي دائماً حلاً أمثل.

ب. استعمل غابة المجموعات المنفصلة السريعة المشروحة في المقطع 3.12 لتحسين الخوارزمية بفعالية. افترض أن مجموعة مهام الدخل، مرتبة سابقاً بحسب الترتيب المتناقص باطراد للغرامات. حلّل زمن تنفيذ خوارزمتك.

#### 5-16 التحية المفصلة عن الخط

نستخدم الحواسيب الحديثة عنايةً لحزن كميات صغيرة من المعطيات في ذاكرة سريعة. ومع أن برنامجاً ما قد يُنقذ إلى كميات كبيرة من المعطيات، إلا أن عززَ مجموعة جزئية صغيرة من الذاكرة الرئيسة في *cache* - وهي ذاكرة صغيرة ولكنها أسرع - يمكن أن يُنقذ زمن النفاذ الكلي تحفيظاً كبيراً. حين يُنقذ برنامجٌ حاسوبي ما فإنه يصنع متاليةً  $(r_1, r_2, \dots, r_n)$  من طلباتِ النفاذ إلى الذاكرة، حيث يكون كل طلب لعنصر معطيات خاص. على سبيل المثال، يمكن لبرنامج يُنقذ إلى أربعة عناصر متمايزة  $(a, b, c, d)$  أن يصنع متالية الطلبات  $(d, b, d, b, d, a, c, d, b, a, c, b)$ . ليكن  $k$  حجم الحاية. حين تتضمن الحاية  $k$  عنصراً، ويطلب البرنامج العنصر ذا الترتيب  $k + 1$ ، فيجب أن يقرّر النظام، لهذا الطلب ولكل طلبٍ تالي، ما هي العناصر الـ  $k$  التي يجب أن يحتفظ بها في الحاية. وبعبارة أدق، تتحقق خوارزمية إدارة الحاية من وجود العنصر  $r_i$  سابقاً في الذاكرة، وذلك لكل طلبٍ  $r_i$ . فإذا كان موجوداً فلدينا *إصابة الحاية cache-hit*، وإلا فلدينا *عدم إصابة الحاية cache-miss*. في حالة عدم الإصابة، يسحب النظام العنصر  $r_i$  من الذاكرة الرئيسة، وعلى خوارزمية إدارة الحاية أن تقرر إما الاحتفاظ بالعنصر  $r_i$  في الحاية وإما عدم الاحتفاظ. فإذا قرّرت الاحتفاظ بالعنصر  $r_i$ ، وكان في الحاية  $k$  عنصراً، وجب أن تطرد عنصراً لتخلي مكاناً لـ  $r_i$ . تطرد خوارزمية إدارة الحاية عناصر بمدف تصغير عدد مرات عدم إصابة الحاية على كامل متالية الطلبات.

إن مسألة الحاية هي عادة مسألة على الخط *on-line*. أي علينا أن نتخذ قراراً بشأن المعطيات التي ينبغي أن تبقى في الحاية دون أن نعلم الطلبات المستقبلية. ولكننا هنا ندرس النسخة المفصلة عن الخط *off-line* من هذه الخوارزمية، حيث لدينا سلفاً متالية الطلبات كاملة (وعددها  $n$  طلباً) وحجم الحاية  $k$ ، ونريد أن نجعل العدد الكلي لحالات عدم الإصابة أصغر.

يمكننا حل هذه المسألة المفصلة عن الخط باستخدام استراتيجية شرهة تسمى **الأبعد في المستقبل** *further-in-future*، حيث تختار طرد البند الموجود في الخاية الذي يكون نفاذه القادم في متتالية طلبات النفاذ الأبعد في المستقبل.

- أ. اكتب شبه رماز للمدير خاية يستعمل استراتيجية الأبعد في المستقبل. ينبغي أن يكون الدخول متتالية الطلبات  $(r_1, r_2, \dots, r_n)$  وحجم الخاية  $k$ ، وينبغي أن يكون الخرج متتالية القرارات المتعلقة بعناصر للمعطيات التي يجب إدخالها عند كل طلب (إن وجدت). ما هو زمن تنفيذ هذه الخوارزمية؟
- ب. بين أن مسألة الفخية المفصلة عن الخط تُظهر بنية جزئية مُثلّية.
- ت. بين أن إجراء الأبعد في المستقبل يُنتج أصغر عدد ممكن من حالات عدم إصابة الخاية.

## ملاحظات الفصل

- يمكن أن نجد لهذه عن الخوارزميات الشرهة والكيانات المصفوفية في Lawler [224] و Papadimitriou و Steiglitz [271].
- ظهرت الخوارزمية الشرهة أول مرة في كتب الأمثلة التوافقية في عام 1971 في مقال لـ Edmonds [101]، علماً بأن نظرية الكيانات المصفوفية تعود إلى تاريخ سابق في عام 1935 في مقال لـ Whitney [355].
- يعتمد برهاننا على صحة الخوارزمية الشرهة في مسألة اختيار النشاطات على المرجع Gavril [131].
- ونجد دراسة مسالية جدولة للمهام في Lawler [224]؛ Horowitz و Sahni و Rajasekaran [181] و Brassard و Bratley [55].
- اشتهرت أزمته Huffman في عام 1952 [185]، وقد استكشف Lelewer و Hirschberg [231] تقنيات ضغط المعطيات التي كانت معروفة حتى عام 1987.
- أما توسيع نطاق نظرية الكيان للمصفوفي إلى نظرية الكيان الشره، فقد كان رائداها Lovász و Korte [216, 217, 218, 219]، اللذين عُمما النظرية التي قدمناها هنا.

## 17 تحليل الكلفة المخمّدة

عند إجراء تحليل الكلفة المخمّدة *amortized analysis*، نحسب متوسط الزمن اللازم للقيام بمتتالية من العمليات على بنية معطيات ما. يمكن استخدام هذا التحليل لبيان أن متوسط كلفة عملية ما صغير عندما نقوم بحساب المتوسط على متتالية من العمليات، حتى لو كانت عملية واحدة من بين هذه العمليات مكلفة. يختلف تحليل الكلفة للمختمدة عن تحليل الحالة الوسطى في أنه لا يستخدم الاحتمالات؛ فتحليل الكلفة للمختمدة يعطي ضماناً على الأداء المتوسط لكل عملية في أسوأ الحالات.

نعالج المقاطع الثلاثة الأولى من هذا الفصل التفصيلات الثلاث الأكثر انتشاراً في تحليل الكلفة المخمّدة. إذ يبدأ المقطع 1.17 بالتحليل المجمع *aggregate analysis*، الذي نحدّد باستخدامه حدّاً أعلى  $T(n)$  للكلفة الكلية لمتتالية من  $n$  عملية. وبذلك يكون متوسط كلفة العملية الواحدة هو  $T(n)/n$ . ونعتبر أن الكلفة المتوسطة هي الكلفة المخمّدة لكل عملية، وبهذا يكون لكل العمليات الكلفة المخمّدة نفسها.

ونتناول المقطع 2.17 طريقة المحاسبة، التي نحدّد فيها كلفة مختمدة لكل عملية، وعندما يكون هناك أكثر من غلط من العمليات، يكون لكل غلط كلفة مختمدة مختلفة. تحمّل طريقة المحاسبة بعض العمليات في بداية متتالية العمليات كلفة إضافية، ونحوّل هذه الكلفة الإضافية كـ "رصيد مسبق الدفع" لغرضي محدّد في بنية المعطيات. يُستخدم هذا الرصيد لاحقاً للتسديد عن عمليات بأن يُطلب منها أقل مما تكلف فعلاً.

يناقش المقطع 3.17 طريقة الكمون، التي نحدّد - بطريقة مشابهة لطريقة المحاسبة - الكلفة المختمدة لكل عملية، وقد نحمل بعض العمليات المبكرة أكثر من كلفتها لنعوّض عن تحميل عمليات بأقل من الكلفة لاحقاً. نحفظ طريقة الكمون بقيمة اعتماد على أنها "الطاقة الكامنة" لبنية المعطيات ككل، بدلاً من ربط الاعتماد بأغراض فردية داخل بنية المعطيات.

سنستخدم مثالين ندرس من خلالها هذه الطرق الثلاث. الأول هو ممكن مع عملية إضافية، هي MULTIPOP، التي تُنزع عدّة أغراض من المكبس دفعةً واحدة. والمثال الثاني هو عداد التائي بعدد بدءاً من 1 بالاعتماد على عملية وحيدة هي INCREMENT.

تذكّر، وأنت تقرأ هذا الفصل، أن المحاولات للسندة خلال عملية تحليل الكلفة للمختمدة ما هي إلا هدف التحليل فقط، ولا ضرورة لظهورها في الرماز، بل ينبغي ألا تظهر فيه. فمثلاً، إذا أسند اعتماداً إلى

غرضي ما  $x$  عند استخدام طريقة المحاسبة، فليست هناك حاجة إلى إسناد مقدار مناسب إلى واصفة ما - مثل  $x.credit -$  في الرماز.

عندما نقوم بتحليل الكلفة المخبّدة لبنية معطيات محددة، فإننا غالبًا ما نكسب نظرة أعمق قد تساعد في تحسين التصميم. ونستخدم مثالًا في المقطع 4.17، طريقة الكمون لتحلّل جدولاً يتمدّد ويتقلّص ديناميكيًا.

## 1.17 التحليل المخبّع

نبيّن، في التحليل المخبّع *aggregate analysis*، أن متاليّة من  $n$  عملية تستغرق زمنًا كليًا في أسوأ الحالات هو  $T(n)$ ، لكل قيم  $n$ . إن متوسط كلفة العملية الواحدة، أو كلفتها المخبّدة *amortized cost*، في أسوأ الحالات هو  $T(n)/n$ . لاحظ أن هذه الكلفة للمخبّدة تنطبق على كلّ عملية، حتى عندما يكون هناك عدة أنواع من العمليات في المتاليّة. أما فيما يخص الطريقتين التاليتين اللتين سندرسهما في هذا الفصل - طريقة المحاسبة وطريقة الكمون - فإنهما قد تستندان كلًّا منهما مخبّدة مختلفة إلى الأنواع المختلفة من العمليات.

### عمليات المكس

ندرس في مثالنا الأول المتعلق بالتحليل المخبّع كدساتٍ أضفنا إليها عملية جديدة. عرضنا في المقطع 1.10 عمليّتي المكس الأساسيتين، وذكرنا أن كلّاً منهما يستغرق زمنًا  $O(1)$ ، وهما:

$PUSH(S, x)$  تدفع الفرض  $x$  إلى للمكس  $S$ .

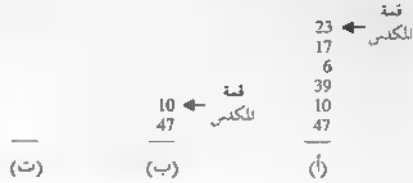
$POP(S)$  تزعّج الفرض من أعلى للمكس  $S$ ، وتعيده. وإن استدعاء عملية  $POP$  على مكس فارغ يُحدّث خطأً.

ولما كانت كلتا العمليتين تُنفّذان في زمن  $O(1)$ ، فسنفترض أن كلفة كلّ منها هي 1. وبذلك تكون الكلفة الكلية لمتاليّة من  $n$  عملية  $PUSH$  و  $POP$  هي  $n$ ، ويكون زمن التنفيذ الفعلي لـ  $n$  عملية هو  $\Theta(n)$ .

نضيف الآن عملية للمكس  $MULTIPOP(S, k)$ ، التي تُخرّج  $k$  غرضًا من قمة للمكس أو تفرغه كله إذا كان يحوي أغراضًا عددها أقل من  $k$  غرضًا. نفترض طبعا أن  $k$  موجب، وإلا فإن عملية  $MULTIPOP$  لا تُحدّث أيّ تغيير في للمكس. في شبه الرماز التالي، تعيد العملية  $STACK-EMPTY$  القيمة  $TRUE$  إذا لم يكن هناك أيّ غرض في للمكس عند استدعائها، و  $FALSE$  فيما عدا ذلك.

$MULTIPOP(S, k)$

```
1 while not STACK-EMPTY(S) and k > 0
2   POP(S)
3   k = k - 1
```



**الشكل 1.17** أثر عمل MULTIPOP على مكس  $S$  مَبْنٍ بدايةً في (أ). نَزَعُ عمليةً MULTIPOP( $S, 4$ ) الأغراض الأربعة في أعلى المكس فيصبح المكس كما في (ب). العملية التالية هي MULTIPOP( $S, 7$ ) وهي تفرغ المكس - كما هو مبين في (ت) - لأن ما بقي فيه أقل من 7 أغراض.

يُؤَيِّن الشكل 1.17 مثالاً على MULTIPOP.

ما هو زمن تنفيذ MULTIPOP( $S, k$ ) على مكس فيه  $s$  غرضاً؟ إن زمن التنفيذ الفعلي خطيٌ بدلالة عدد عمليات POP المنفذة فعلياً، ولهذا السبب يكفي أن نحلل MULTIPOP باستخدام الكلفة المجرّدة  $I$  لكل عملية PUSH وعملية POP. إن عدد التكرارات في حلقة **While** هو  $\min(s, k)$  غرضاً مَزْوَعاً من المكس. وفي كل تكرارٍ للحلقة يحدث استدعاءٌ واحد لعملية POP في السطر 2. إذن الكلفة الكلية لـ MULTIPOP هو  $\min(s, k)$ ، وزمن التنفيذ الفعلي هو دالة خطية لهذه الكلفة.

دعونا ندرس متتالية من  $n$  عملية PUSH و POP و MULTIPOP على مكس خالٍ بدايةً. إن كلفة عملية MULTIPOP في المتتالية هي  $O(n)$  في أسوأ الحالات، وذلك لأن حجم المكس هو على الأكثر  $n$ . لذا، فإن زمن أية عملية على المكس هي في أسوأ الحالات  $O(n)$ ، ونتيجةً لذلك، فإن كلفة  $n$  عملية هو  $O(n^2)$ ، لأنه قد يكون لدينا  $O(n)$  عملية MULTIPOP، كلُّها منها  $O(n)$ . ومع أن هذا التحليل صحيح، فإن النتيجة التي يعطيها  $O(n^2)$ ، بأن يأخذ كلفة أسوأ الحالات لكل عملية على حدة ليست بحكمة كافية.

يمكننا، باستخدام التحليل المتشعب، الحصول على حدٍّ أعلى أفضل يأخذ بالاعتبار مجمل متتالية العمليات، وعددها  $n$ . والواقع أنه على الرغم من أن عملية MULTIPOP واحدة قد تكون مكلفة، فإن أية متتالية من  $n$  عملية PUSH و POP و MULTIPOP على مكس خالٍ بدايةً تكلف على الأكثر  $O(n)$ . لماذا؟ لأن كلَّ غرضٍ يمكن أن يُنَزَع مرةً واحدة على الأكثر لكل مرة يُدفع به إلى المكس. لهذا السبب فإن عدد المرات التي يمكن فيها استدعاء POP على مكس غير خالٍ، ومن ضمنها الاستدعاءات داخل MULTIPOP، تساوي على الأكثر عدد عمليات PUSH، وهي على الأكثر  $n$ . أي إن أية متتالية من  $n$  عملية PUSH و POP و MULTIPOP تستغرق في المجمل زمنًا  $O(n)$ ، مهما تكن قيمة  $n$ . ويكون متوسط كلفة عملية هو  $O(n)/n = O(1)$ . في التحليل للمدمج، تسند لكلَّ عملية كلفةً مخنّدةً هي الكلفة المتوسطة، إذن، في هذا المثال تكون الكلفة المخفضة لكلَّ من عمليات المكس الثلاث هي  $O(1)$ .

نؤكد ثانية أنه على الرغم من أننا بينا فقط أن الكلفة المتوسطة، ومن ثم زمن التنفيذ، لعملية مكس هي  $O(1)$ ، إلا أننا لم نستخدم أية محاكمة احتمالية. فقد عرضنا فعلياً حلاً في أسوأ الحالات،  $O(n)$ ، على متتالية من  $n$  عملية. وتقسيم هذه الكلفة الكلية على  $n$ ، حصلنا على الكلفة المتوسطة للعملية الواحدة، أو ما يُعرف بكلفتها المحققة.

### زيادة عدد الثاني

لندرس مثالاً آخر للتحليل المجمع، وهو مسألة تخزين عدد اثنائي من  $k$  بتاً يُقَدُّ تصاعدياً بدءاً من 0. نستخدم، لتمثيل العداد صيغة بتات  $A[0 \dots k-1]$ ، حيث  $A.length = k$ . فإذا كان العدد الاثنائي  $x$  مخزناً في العداد، فإن البت الأقل مرتبة منه يكون مخزناً في  $A[0]$ ، فيما يُخزَّن البت الأعلى مرتبة في  $A[k-1]$ ، بحيث يكون  $x = \sum_{i=0}^{k-1} A[i] \cdot 2^i$ . في البداية يكون  $x = 0$ ، ومن ثم فإن  $A[i] = 0$  لكل القيم  $i = 0, 1, \dots, k-1$ . ولإضافة 1 (بالمقاس  $2^k$ ) إلى القيمة في العداد، فإننا نستخدم الإجراء التالي.

INCREMENT(A)

```

1  i = 0
2  while i < A.length and A[i] == 1
3      A[i] = 0
4      i = i + 1
5  if i < A.length
6      A[i] = 1

```

يبيّن الشكل 2.17 ما يحدث لعداد اثنائي عندما نُزيد 16 مرة، بدءاً من القيمة البدائية 0، وانتهاءً بالقيمة 16. مع بداية كل تكرار في حلقة while في السطور 2-4، نريد أن نضيف 1 في الموقع  $i$ . فإذا كان  $A[i] = 1$ ، فإن زيادة 1 تقلب البت في الموقع  $i$  إلى 0، وتعطي خلاً قدره 1 لإضافته إلى الموقع  $i + 1$  في التكرار التالي للحلقة. وإلا فإن الحلقة تتوقف، فإذا كان  $k < i$ ، فإننا نعلم أن  $A[i] = 0$ ، ولذلك فإن إضافة 1 في الموقع  $i$  يؤدي إلى قلب 0 إلى 1، وهذا ما يتكفل به السطر 6. إن كلفة كل عملية INCREMENT خطية مع عدد البتات التي تتعرض للقلب.

وكما في مثال المكس، يعطينا تحليل سريع cursory حلاً صحيحاً إلا أنه غير محكم كفاية. وإن تنفيذاً واحداً لـ INCREMENT يستغرق زمناً  $\Theta(n)$  في أسوأ الحالات، وذلك عندما نحوي الصفيفة  $A$  القيمة 1 في كل المواقع. إذن، فإن متتالية من  $n$  عملية INCREMENT على عداد قيمته 0 بدايةً يستغرق زمناً  $O(nk)$  في أسوأ الحالات.

يمكننا أن نجعل تحليلنا أكثر دقةً ليعطي كلفةً في أسوأ الحالات  $O(n)$  لمتتالية من  $n$  عملية INCREMENT، وذلك بملاحظة أنه لا تُقلب كل البتات عند كل استدعاء لـ INCREMENT. وكما يبيّن

قيمة العداد	$A[7]$	$A[6]$	$A[5]$	$A[4]$	$A[3]$	$A[2]$	$A[1]$	$A[0]$	الكلفة الكليّة
■	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1
2	0	0	0	0	0	0	1	0	3
3	0	0	0	0	0	0	1	1	4
4	0	0	0	0	1	0	0	0	7
5	0	0	0	0	1	0	1	0	8
6	0	0	0	0	1	1	0	0	10
7	0	0	0	0	1	1	1	0	11
■	0	0	0	1	0	0	0	0	15
9	0	0	0	1	0	0	1	0	16
■	0	0	0	1	0	1	0	0	18
11	0	0	0	1	0	1	1	0	19
12	0	0	0	1	1	0	0	0	22
13	0	0	0	1	1	0	0	0	23
14	0	0	0	1	1	1	0	0	25
15	0	0	0	1	1	1	1	0	26
16	0	0	0	1	0	0	0	0	31

**الشكل 2.17** عَدَد الثاني من 8 بتات تزايد قيمته من 0 إلى 16 باستدعاء متتالية من 16 عملية INCREMENT. إن البتات التي تنقلب لأخذ القيمة التالية مظللة. وكلفة التنفيذ المقابلة لقلب البتات مبيّنة إلى يمين العداد. لاحظ أن الكلفة الكليّة لا تتجاوز أبدًا ضعف عدد عمليات INCREMENT.

الشكل 2.17، فإن  $A[0]$  ينقلب عند كل استدعاء لـ INCREMENT. أما البت التالي له  $A[1]$ ، فإنه ينقلب مرة في كل استدعاءين؛ أي إن متتاليّة من  $n$  عملية INCREMENT على عَدَد معدوم بدايةً تتسبب في قلب  $A[1]$   $\lfloor n/2 \rfloor$  مرة. وبالمثل، ينقلب البت  $A[2]$  مرة كل أربع مرات، أو  $\lfloor n/4 \rfloor$  مرة في متتالية من  $n$  عملية INCREMENT. وبوجه عام، عندما  $i = 0, 1, \dots, k-1$ ، ينقلب البت  $A[i]$  فقط  $\lfloor n/2^i \rfloor$  مرة في متتالية من  $n$  عملية INCREMENT على عَدَد معدوم بدايةً. وإذا كان  $i \geq k$ ، فإن البت  $A[i]$  غير موجود أصلاً، وبالتالي لا يمكن له أن ينقلب. إذن، فالعدد الكلي للقلبات في المتتالية هو

$$\sum_{i=0}^{k-1} \lfloor \frac{n}{2^i} \rfloor < n \sum_{i=0}^{\infty} \frac{1}{2^i} \\ = 2n ,$$

وذلك باستخدام المعادلة (أ.6). ونتيجةً لذلك، يكون زمن تنفيذ متتالية من  $n$  عملية INCREMENT على عَدَد معدوم بدايةً هو  $O(n)$  في أسوأ الحالات. ومن ثم تكون الكلفة المختدة للعملية الواحدة هي  $O(n)/n = O(1)$ .

## تعاريف

### I-1.17

إذا تضاعفت مجموعة عمليات المكس عملية MULTIPUSH تدفع  $k$  عنصرًا في المكس، هل سيبقى الكلفة

المختدة لأية عملية مكس محدة بالحد (1)؟

2-1.17

بيّن أنه إذا تم تضمين عملية DECREMENT في مثال العدد ذي  $k$  بتاً، فإن  $n$  عملية قد تكلف ما قد يصل إلى زمن  $O(nk)$ .

3-1.17

افترض أننا ننفذ متتالية من  $n$  عملية على بنية معطيات ما. تكلف العملية ذات الرقم  $i$  الكلفة  $i$  إذا كانت  $i$  إحدى قوى 2، وتكلف 1 في باقي الحالات. استخدم التحليل المجمّع لتحديد الكلفة المختدة للعملية.

## 2.17 طريقة المحاسبة

نستد، في طريقة المحاسبة *accounting method* للتحليل المختد، كلفاً مختلفاً للعمليات المختلة، مع تحميل بعض العمليات أكثر من كلفتها الفعلية أو أقل منها. تُسمى الكلفة المحملة لعملية ما **كلفتها المختدة** *amortized cost*. عندما تتجاوز الكلفة المختدة لعملية ما كلفتها الفعلية، يُسند الفارق إلى أغراض معينة في بنية المعطيات كـ **رصيد credit**. قد يساعد هذا الرصيد لاحقاً في التسديد لمصلحة عمليات تكون كلفتها المختدة أقل من كلفتها الفعلية. إذن، يمكن أن نرى الكلفة المختدة لعملية ما على أنها مفرقة إلى كلفة فعلية ورصيد إما أن يكون مودعاً أو مستهلكاً. قد يكون للعمليات المختلة كلفاً مختدة متباينة، وبهذا تختلف هذه الطريقة عن التحليل المجمّع في أن لكل العمليات الكلفة المختدة نفسها.

يجب أن نختار الكلف المختدة بعناية. فإذا كنا نريد تحليلاً بالكلف المختدة لنبيّن أن متوسط كلفة العملية الواحدة في أسوأ الحالات صغير، فعلياً أن نتحقق من أن الكلفة الكلية المختدة لمتتالية من العمليات تمثل حداً أعلى على الكلفة الكلية الفعلية لهذه المتتالية. إضافة إلى ذلك، وكما هو الحال في التحليل المجمّع، يجب أن تكون هذه العلاقة محققة على كل متتاليات العمليات. إذا أشرنا إلى الكلفة الفعلية للعملية  $i$  بـ  $c_i$ ، وإلى كلفتها المختدة بـ  $e_i$ ، فإننا نشترط أن نحقق كل للنتاليات المكونة من  $n$  عملية للتراجحة التالية:

$$\sum_{i=1}^n e_i \geq \sum_{i=1}^n c_i \quad (1.17)$$

إن الرصيد الكلي *total credit* المخزون في بنية المعطيات هو الفارق بين الكلفة الكلية المختدة والكلفة الفعلية، أي  $\sum_{i=1}^n e_i - \sum_{i=1}^n c_i$ . واعتماداً على للتراجحة (1.17)، يجب أن يكون الرصيد الكلي الموافق لبنية المعطيات موجباً على الدوام. وفي حال سُحِبَ للرصيد الكلي أن يصبح سالباً (نتيجة تحميل أولى العمليات أقل من كلفتها مع الوعد بتسديد الحساب لاحقاً)، فستكون الكلف الكلية المختدة المحققة في حينها أقل من الكلف الكلية الفعلية المحققة؛ وعندها لن تكون الكلفة الكلية المختدة فيما يخص متتالية العمليات حتى تلك



الملاحظة، حدًا أعلى على الكلفة الكلية الفعلية. إذن، يجب أن نغير اهتمامنا كي لا يصبح الرصيد الكلي في بنية المعطيات سالبًا أبدًا.

### عمليات المكس

لإيضاح طريقة المحاسبة للتحليل للمخمد، سنعود إلى مثال للمكس. تذكر أن الكلفة الفعلية للعمليات كانت:

PUSH 1 ,  
POP 1 ,  
MULTIPOP  $\min(k, s)$  ,

حيث  $k$  هي المخمد للمعطى لـ MULTIPOP، و  $s$  حجم المكس عند استدعائها. دعنا نحدد الكلف للمختدة التالية:

PUSH 2 ,  
POP 0 ,  
MULTIPOP ■ .

لاحظ أن الكلفة للمختدة لـ MULTIPOP ثابتة (0)، فيما كلفتها الفعلية متغيرة. في هذه الحالة، كل الكلف للمختدة الثلاث ذات قيم ثابتة، مع أن الكلف للمختدة للعمليات المدروسة قد تتباين عمومًا بالمقارنة.

سنرى الآن أن بمقدورنا التسديد لأية متالية من عمليات المكس بتحميل الكلف للمختدة. افترض أننا نستخدم دولارًا لتمثيل كل وحدة من وحدات الكلفة. نبدأ بمكسي فارغ. تذكر التشابه الذي ذكرناه في المقطع 1.10 بين مكس بنية المعطيات ومكس الأطباق في كافيتريا. عندما نضيف طبقًا إلى المكس، فإننا ندفع دولارًا لتسديد الكلفة الفعلية لهذه الإضافة، ويبقى معنا رصيد من دولار واحد (من الدولارين المدفوعين)، نضعه في الطبق. في أية لحظة، كل طبق في المكس يحوي رصيدًا قدره دولار واحد.

إن الدولار المخزن في الطبق هو تسديد مسبق لكلفة سحب الطبق من المكس. وعندما ننفذ عملية POP، فإننا لا نحمل العملية أية كلفة، ونسدد كلفتها الفعلية باستخدام الرصيد المخزن في المكس. ولكي نسحب طبقًا، نأخذ دولار الرصيد من الطبق ونستخدمه للتسديد لعملية السحب. إذن بتحميل عملية PUSH أكثر قليلًا، لا نضطر إلى تحميل عملية POP أية كلفة.

إضافة إلى ذلك، يمكننا ألا نحمل عمليات MULTIPOP أية كلفة. فلكي نسحب الطبق الأول، نأخذ دولار الرصيد من الطبق ونستخدمه لتسديد الكلفة الفعلية لعملية POP، وهكذا دواليك... ولتسحب الطبق الثاني، نأخذ ثانية دولار الرصيد من الطبق لتسديد الكلفة الفعلية لعملية POP. وهكذا نكون قد حوّلنا على الدوام مقدّمًا قدرًا كافيًا لتسديد عمليات MULTIPOP. وبعبارة أخرى، لما كان كل طبق من المكس يحوي رصيدًا من دولار واحد، ولما كان للمكس يحوي عددًا موجبًا من الأطباق، كان لأية متالية من  $n$  عملية MULTIPOP و POP و PUSH كلفة كلية مخمدة هي حد أعلى على الكلفة الكلية الفعلية. ولما كانت الكلفة

الكلية المعتمدة من المراتبة  $O(n)$ ، فهذه هي أيضًا حال الكلفة الكلية الفعلية.

### زيادة عدد التاني

ثمّة إيضاح آخر لطريقة المحاسبة يتمثل في تحليل عملية INCREMENT على عدد اثنائي يبدأ من الصفر. إن زمن تنفيذ هذه العملية، كما لاحظنا سابقًا، متناسب مع عدد البتات المقلوبة، وهذا ما سنستخدمه بوصفه الكلفة في هذا المثال. وسنستخدم الدولار مرة أخرى لتمثيل واحدة الكلفة (قلب بت في هذا المثال).

للقيام بالتحليل المعتمد، نحمل كلفة محدّدة قدرها دولاران لجعل قيمة البت 1. عندما يأخذ بت القيمة 1 فإننا نستخدم دولارًا (من الدولارين المصطلين) لتسديد الكلفة الفعلية لتغيير قيمة البت، ونضع الدولار الآخر على البت ليكون رصيدًا يُستخدم لاحقًا عندما تقلب البت مجددًا إلى 0. في أية لحظة كل 1 في العداد لديه رصيد من دولار واحد عليه، وهكذا لا نحتاج إلى تحميله أية كلفة لإعادته إلى 0؛ فنحن نسدّد عملية العودة إلى الصفر باستخدام الدولار على البت.

يمكننا الآن تحديد الكلفة المعتمدة لـ INCREMENT. تُسدّد عودة البتات إلى الصفر في حلقة while باستخدام الدولارات على البتات المصفّرة. تفرّز INCREMENT على الأكثر بتًا واحدًا إلى 1 في السطر 6، وهكذا تبلغ الكلفة المعتمدة لعملية INCREMENT على الأكثر دولارين. إن عدد الوجدان في العداد لا يصبح سالبًا أبدًا، وهكذا فإن قيمة الرصيد تبقى دائمًا موجبة. إذن، الكلفة المعتمدة الكلية لـ INCREMENT هي  $O(n)$ ، وهي التي تُعدّ الكلفة الفعلية.

### تعاريف

#### 1-2.17

افترض أننا نُنفّذ متتالية من عمليات مكبس لا يتجاوز حجمه  $k$  أبدًا. بعد كل  $k$  عملية، نُعيدُ نسخة عن كامل المكبس لأغراض الحزن الاحتياطي. يبيّن أن كلفة  $n$  عملية مكبس - ومن ضمنها نسخ المكبس - هي  $O(n)$ ، وذلك بإسناد الكلف المعتمدة المناسبة لمختلف العمليات لتنفيذ على المكبس.

#### 2-2.17

أعدّ التصرين 1.17-3 باستخدام التحليل بطريقة المحاسبة.

#### 3-2.17

افترض أننا لا نرغب في زيادة عداد ما فقط، بل بإعادته إلى الصفر أيضًا (أي يجعل كل بتاته تساوي 0). وبافتراض أن الزمن اللازم لقراءة بت أو تغييره هو  $\Theta(1)$ ، يبيّن كيف يمكن تنجز عداد كصفيغة من البتات بحيث تستغرق أية متتالية من  $n$  عملية INCREMENT و RESET زمنًا  $O(n)$ ، وذلك على عداد معدوم بدايةً. (لمصيح: احتفظ بمؤشر إلى البت 1 ذي المراتبة الأعلى.)

## 3.17 طريقة الكمون

بدلاً من تمثيل العمل المستد سلفاً كرسيد عجز في أغراض محددة في بنية المعطيات، تمثل طريقة الكمون *potential method* التحليل للمختد العمل المستد سلفاً "كطاقة كامنة" أو فقط "كمون" يمكن تحريره لتسديد عمليات مستقبلية. يُربط الكمون ببنية للمعطيات ككل بدلاً من ربطه بأغراض محددة داخل البنية. تعمل طريقة الكمون كالتالي. نُشير  $\Phi$  بعملية، متدئين ببنية معطيات بدائية  $D_0$ . ولكن  $c_i$  الكلفة الفعلية للعملية ذات الرقم  $i$ ، لكل  $i = 1, 2, \dots, n$ . ولكن  $D_i$  بنية للمعطيات التي تنتج بعد تطبيق العملية ذات الرقم  $i$  على بنية المعطيات  $D_{i-1}$ . تقابل دالة كمون *Potential function*  $\Phi$  كل بنية معطيات  $D_i$  بعدد حقيقي  $\Phi(D_i)$ ، هو الكمون المرافق لبنية للمعطيات  $D_i$ . فتكون  $c_i$  الكلفة المختدة *amortized cost* للعملية ذات الرقم  $i$ ، بالاعتماد على دالة الكمون  $\Phi$  معرفة بـ

$$c_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) . \quad (2.17)$$

إذن، الكلفة المختدة لكل عملية هي كلفتها الفعلية مضافاً إليها الزيادة في الكمون الناتج عن هذه العملية. واعتماداً على المعادلة (2.17)، تكون الكلفة الكلية للمختدة لـ  $n$  عملية:

$$\begin{aligned} \sum_{i=1}^n c_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0) . \end{aligned} \quad (3.17)$$

نتبع المساواة الثانية من المعادلة (9.أ)، وذلك لأن الحدود  $\Phi(D_i)$  تُحذف فيما بينها.

إذا استعنا تعريف دالة كمون  $\Phi$  بحيث يكون  $\Phi(D_n) \geq \Phi(D_0)$ ، فإن الكلفة الكلية للمختدة  $\sum_{i=1}^n c_i$  تعطي حداً أعلى على الكلفة الكلية الفعلية  $\sum_{i=1}^n c_i$ . عملياً، لا نعرف دائماً عدد العمليات التي يمكن القيام بها. ولذا إذا افترضنا  $\Phi(D_i) \geq \Phi(D_0)$  لكل قيم  $i$ ، فإننا نضمن، كما في طريقة المحاسبة، أن نستد سلفاً. نكتفي عادة بتعريف  $\Phi(D_0)$  على أنه 0، ثم نبين أن  $\Phi(D_i) \geq 0$  لكل قيم  $i$ . (انظر التمرين 1-3.17 للاطلاع على طريقة سهلة لمعالجة الحالات التي يكون فيها  $\Phi(D_0) \neq 0$ ).

نرى حديثاً أنه إذا كان فرق الكمون  $\Phi(D_i) - \Phi(D_{i-1})$  للعملية  $i$  موجباً، كانت الكلفة المختدة  $c_i$  تمثل تحميلاً زائداً للعملية  $i$ ، وكان كمون بنية للمعطيات في تزايد. وإذا كان فرق الكمون سالباً، فإن الكلفة المختدة تمثل تحميلاً أقل من اللازم للعملية  $i$ ، وتستد الكلفة الفعلية للعملية بإنقاص الكمون.

تعتمد الكلف المختدة للمعرفة في المعادلتين (2.17) و (3.17) على اختيار دالة الكمون  $\Phi$ . إذ قد تُنتج دوال كمون مختلفة كلفاً مختدة مختلفة، إلا أنها كلها حدوداً عليا للكلف الفعلية. في كثير من الأحيان هناك تسويات يمكن تحقيقها عند اختيار دالة الكمون؛ تعتمد أفضل دالة كمون يمكن استخدامها على حدود الزمن المرغوبة.

### عمليات المكس

لتوضيح طريقة الكمون، نعود مرة ثانية إلى مثال عمليات للمكس PUSH و POP و MULTIPOP. نعرف دالة الكمون  $\Phi$  على مكس على أنها عدد الأغراض في المكس. إن دالة الكمون للمكس الفارغ  $D_0$  الذي نبدأ به هي  $\Phi(D_0) = 0$ . ولما كان من غير الممكن أن يكون عدد الأغراض في المكس سالبًا، فإن للمكس  $D_i$  الناتجة بعد العملية  $i$  كمونًا غير سالب، وهكذا فإن

$$\begin{aligned}\Phi(D_i) &\geq 0 \\ &= \Phi(D_0) .\end{aligned}$$

تمثل الكلفة الكلية للمخدمة لـ  $n$  عملية بالاعتماد على  $\Phi$  إذن حدًا أعلى على الكلفة الفعلية. لنحسب الآن الكلف المخدمة لمختلف عمليات المكس. إذا كانت العملية ذات الرقم  $i$  على مكس بحوي  $s$  غرضًا هي عملية PUSH، فإن فرق الكمون هو

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &= (s + 1) - s \\ &= 1 .\end{aligned}$$

واعتمادًا على المعادلة (2.17) فإن الكلفة للمخدمة لعملية PUSH هذه هي

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= 1 + 1 \\ &= 2 .\end{aligned}$$

لنفترض أن العملية ذات الرقم  $i$  على المكس هي MULTIPOP( $S, k$ )، وأن  $k' = \min(k, s)$  هو عدد الأغراض المدفوعة خارج المكس. إن الكلفة الفعلية للعملية هي  $k'$ ، وفرق الكمون هو:

$$\Phi(D_i) - \Phi(D_{i-1}) = -k' .$$

إذن، الكلفة المخدمة لعملية MULTIPOP هي

$$\begin{aligned}\hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &= k' - k' \\ &= 0 .\end{aligned}$$

وبالمثل، فإن الكلفة للمخدمة لعملية POP اعتيادية هي 0.

إن الكلفة المخدمة لكل من العمليات الثلاث هي  $O(1)$ ، وهكذا فإن الكلفة الكلية المخدمة لمتتالية من  $n$  عملية هي  $O(n)$ . ولما كنا قد بينّا قبل قليل أن  $\Phi(D_i) \geq \Phi(D_0)$ ، فإن الكلفة الكلية للمخدمة لـ  $n$  عملية تمثل حدًا أعلى للكلفة الكلية الفعلية. وبهذا تكون الكلفة في أسوأ الحالات لـ  $n$  عملية  $O(n)$ .

### زيادة عدد الثاني

سندرس مثالاً آخر لطريقة الكمون، وذلك بأن ننظر ثانية إلى عملية زيادة عدد الثاني. نعرف هذه المرة كمون

المعدّد  $b_i$  بعد عملية INCREMENT ذات الرقم  $i$ ، بأنه عدد الخانات ذات القيمة 1 في العداد بعد العملية ذات الرقم  $i$ .

وحساب الكلفة للمخمدة لعملية INCREMENT، نفترض أن عملية INCREMENT ذات الرقم  $i$  تُصغّر  $t_i$  بثًا. فالكلفة الفعلية للعملية هي إذن  $t_i + 1$  على الأكثر، لأنه إضافة إلى تصغير  $t_i$  بثًا، فإن العملية تضع 1 على الأكثر في بت واحد. فإذا كان  $b_i = 0$ ، فهذا يعني أن العملية ذات الرقم  $i$  تُصغّر كل البتات وعددها  $k$ ، ويكون  $b_{i-1} = t_i = k$ . وإذا كان  $b_i > 0$ ، فإن  $b_i = b_{i-1} - t_i + 1$  وفي كلتا الحالتين، تتحقق المتراجحة  $b_i \leq b_{i-1} - t_i + 1$  ويكون فرق الكمون

$$\begin{aligned}\Phi(D_i) - \Phi(D_{i-1}) &\leq (b_{i-1} - t_i + 1) - b_{i-1} \\ &= 1 - t_i.\end{aligned}$$

وعلى هذا تكون الكلفة المعقدة

$$\begin{aligned}C_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\ &\leq (t_i + 1) + (1 - t_i) \\ &= 2.\end{aligned}$$

إذا بدأ المعدّد بالقيمة صفر، فإن  $\Phi(D_0) = 0$ . ولما كان  $\Phi(D_i) \geq 0$  مهما كانت  $i$ ، فإن الكلفة الكلية للمخمدة لمتتالية من  $n$  عملية INCREMENT هي  $O(n)$ .

تعطينا طريقة الكمون أسلوبًا بسيطًا لتحليل العداد، ولو كان لا يبدأ بالصفر. يبدأ العداد بـ  $b_0$  بثًا قيمتها  $i$ ، وبعد  $n$  عملية INCREMENT، يصبح  $b_n$  بثًا قيمتها  $i$  حيث  $b_0 \leq i$  و  $b_n \leq k$ . (تذكّر أن  $k$  هو عدد البتات في المعدّد) يمكننا إعادة كتابة المعادلة (3.17) كالآتي

$$\sum_{i=1}^n c_i = \sum_{i=1}^n C_i - \Phi(D_n) + \Phi(D_0). \quad (4.17)$$

لدينا  $2 \leq C_i$  مهما كانت  $1 \leq i \leq n$ . ولما كان  $\Phi(D_0) = b_0$  و  $\Phi(D_n) = b_n$ ، فإن الكلفة الفعلية لـ  $n$  عملية INCREMENT هي

$$\begin{aligned}\sum_{i=1}^n c_i &\leq \sum_{i=1}^n 2 - b_n + b_0 \\ &= 2n - b_n + b_0.\end{aligned}$$

نلاحظ، بوجه خاص، أن الكلفة الفعلية الكلية هي  $O(n)$ ، مادام  $k = O(n)$ ، وذلك لأن  $b_0 \leq k$ . وبعبارة أخرى، إذا نفّذنا  $n = \Omega(k)$  عملية INCREMENT على الأقل، فإن الكلفة الكلية الفعلية هي  $O(n)$ ، مهما كانت قيمة المعدّد البدائية.

### تمارين

#### 1-3.17

افترض أن لدينا دالة كمون  $\Phi$  بحيث يكون  $\Phi(D_i) \geq \Phi(D_0)$  مهما كانت  $i$ ، إلا أن  $\Phi(D_0) \neq \Phi(D_1)$ . بين أنه توجد دالة كمون  $\Phi'$  بحيث يكون  $\Phi'(D_0) = 0$  و  $\Phi'(D_i) \geq \Phi'$ ، مهما كانت  $i \geq 1$ ، وأن الكلف المضمنة باستخدام  $\Phi'$  هي نفسها الكلف المضمنة باستخدام  $\Phi$ .

#### 2-3.17

أعد التمرين 3-1.17 باستخدام التحليل بطريقة الكمون.

#### 3-3.17

ليكن لدينا بنية معطيات كومة-أصغر ثنائية عادية مع  $n$  عنصرًا تدعم عمليتي INSERT و EXTRACT-MIN بزمن  $O(\lg n)$  في أسوأ الحالات. أعط دالة كمون  $\Phi$  حيث تكون الكلفة المضمنة لعملية INSERT  $O(\lg n)$  والكلفة المضمنة لعملية EXTRACT-MIN هي  $O(1)$ ، وبيّن أن ذلك يعمل كما ينبغي.

#### 4-3.17

ما هي الكلفة الكلية لتنفيذ  $n$  عملية مكس MULTIPOP و POP و PUSH بافتراض أن المكس يبدأ وفيه  $s_0$  غرضًا وينتهي وفيه  $s_n$  غرضًا؟

#### 5-3.17

افترض أن عددًا يبدأ بعدد  $b$  بتأقيمتها  $a$  بتمثيله الاثنائي، عوضًا عن أن يبدأ من 0. بين أن كلفة تنفيذ  $n$  عملية INCREMENT هو  $O(n)$  إذا كان  $n = \Omega(b)$ . (لا تفترض أن  $b$  عدد ثابت.)

#### 6-3.17

بين كيف يمكن تنجز رتل مع عمليتي مكس معادتين (التمرين 6-1.10) بحيث تكون الكلفة المضمنة لكل عملية ENQUEUE وكل عملية DEQUEUE هي  $O(1)$ .

#### 7-3.17

صمّم بنية معطيات تدعم العمليتين التاليتين على مجموعة ديناميكية  $S$  من الأعداد الصحيحة:

INSERT( $S, x$ ) التي تُدخل العنصر  $x$  في  $S$ .

DELETE-LARGER-HALF( $S$ ) التي تحذف من  $S$  أكبر  $\lfloor |S|/2 \rfloor$  عنصرًا.

اشرح كيف يمكن تنجز بنية للمعطيات هذه بحيث تُنفَّذ أية متتالية من  $m$  عملية بزمن  $O(m)$ . ينبغي أن يتضمن تنجزك أيضًا طريقة لإخراج العناصر  $S$  في زمن  $O(|S|)$ .

## 4.17 الجداول الديناميكية

في بعض التطبيقات، لا نعرف سلفاً عدد الأغراض التي ستُخزن في جدول ما؛ فقد نخصص حجمًا لجدول ما، لنكتشف فيما بعد أنه غير كافٍ. في هذه الحالة، يجب إعادة تخصيص حجم أكبر للجدول، ونقل كل الأغراض المخزنة في الجدول الأول ونسخها في الجدول الجديد الأوسع. وبالمثل، إذا خُذفت أغراض كثيرة من الجدول، فقد يكون من المجدى إعادة تخصيص حجم أصغر للجدول. ندرس في هذا المقطع هذه المسألة التي تُسمى بتوسيع جدول ما وتقليصه ديناميكياً. ونبيّن باستخدام التحليل للمحدد أن الكلفة المختدة للإدراج والحذف هي فقط  $O(1)$ ، ولو كانت الكلفة الفعلية لعملية ما كبيرة عندما تطلق توسيماً للجدول أو تقليصاً له. إضافة إلى ذلك، سنرى كيف يمكن ضمان ألا يتجاوز الحيز غير المستخدم في الجدول الديناميكي، أبداً، جزءاً ثابتاً من حجمه الكلي.

نفترض أن الجدول الديناميكي يدعم عمليتي `TABLE-INSERT` و `TABLE-DELETE`. تُدرج عملية `TABLE-INSERT` عنصراً داخل الجدول ليُشغل `slot` واحداً، وهو الحيز المحدد لعنصر واحد. وبالمثل، تُنفّذ العملية `TABLE-DELETE` بحذف عنصر من الجدول، وتحرّر بذلك شقياً. لا أهمية لتفاصيل بنية المعطيات المستخدمة لتنظيم هذا الجدول؛ فقد نستخدم مكدساً (مقطع 1.10)، أو كومة (الفصل 6) أو جدول تليد (الفصل 11). وقد نستخدم أيضاً صفيفة أو مجموعة من الصفيفات لتنجز تخزين الأغراض كما فعلنا في المقطع 3.10.

سنجد من المناسب استخدام مفهوم عزّفته عندما حللنا التليد (الفصل 11). نعرّف عامل التحميل  $\alpha(T)$  لجدول غير خالي  $T$  على أنه عدد العناصر المخزنة في الجدول مقسّمة على حجمه (عدد الشقوب فيه). نسنّد إلى الجدول الخالي (الذي لا يحوي أية عناصر) الحجم 0، ونعرّف عامل تحميله على أنه 1. إذا كان عامل التحميل لجدول ديناميكي محدوداً تحت ثابت ما، فإن الحجم غير المستخدم في الجدول لن يتجاوز أبداً جزءاً ثابتاً من الحجم الكلي.

سنبدأ بتحليل جدول ديناميكي لا ننجز عليه إلا عمليات إدراج. ثم ندرس حالة أكثر عمومية يُسّفع فيها بالقيام بعمليات إدراج وحذف.

## 1.4.17 توسيع الجدول

نفترض أن الجدول يُخزن في صفيفة من الشقوب. وعلّى الجدول عندما تكون كل الشقوب قد استُعملت، أو بصورة مكافئة، عندما يكون عامل تحميله مساوياً 1.<sup>1</sup> في بعض البيئات البرمجية، إذا جُرّزت محاولة لإدراج

<sup>1</sup> قد نرغب في بعض الحالات، عندما يتعلق الأمر مثلاً بجدول تليد مفتوح التعاون، أن نعتبر أن جدولاً ما ملآن عندما يساوي عامل تحميله ثابتاً أصغر من 1 تماماً. (انظر التمرين 14.17)

عنصر في جدول ملاّن، فالبديل الوحيد هو استبعاد المحاولة برسالة خطأ. سنفترض، على أية حال، أن البيئة البرمجية التي تتعامل معها، مثل العديد من البيئات الحديثة، تتيح نظامًا لإدارة الذاكرة قادرًا على تخصيص كتل تخزين وتخزينها عند الطلب. وهكذا عندما يُدرج عنصر في جدول ملاّن، فإن بمقدورنا توسيع *expand* الجدول بتخصيص جدول جديد فيه شقوق أكثر عددًا من الجدول القديم. ولما كنا دائمًا بحاجة إلى أن يكون الجدول مخزنًا في ذاكرة متتالية، كان علينا تخصيص صيغة جديدة للجدول الأكبر ثم نسخ العناصر من الجدول القديم إلى الجدول الجديد.

هناك كسبيّة شائعة *common heuristic* تتمثل في تخصيص جدول جديد فيه ضعف شقوق الجدول القديم. فإذا لم يكن هناك إلا عمليات إدراج، فإن عامل التحميل للجدول  $1/2$  على الأقل دائمًا، وهذا فإن مقدار حجم الذاكرة للمهدور لا يتجاوز أبدًا نصف حجم الجدول.

نفترض، في شبه الرمز التالي، أن  $T$  غرض يمثّل الجدول. يحتوي الواسقة  $T.table$  مؤشرًا إلى كتلة التخزين التي تمثل الجدول، والحقل  $T.num$  عدد العناصر في الجدول، والحقل  $T.size$  عدد الشقوق الكلية في الجدول. في البداية يكون الجدول فارغًا:  $T.num = T.size = 0$ .

#### TABLE-INSERT( $T, x$ )

```

1  if  $T.size == 0$ 
2      allocate  $T.table$  with 1 slot
3       $T.size = 1$ 
4  if  $T.num == T.size$ 
5      allocate new-table with  $2 \cdot T.size$  slots
6      insert all items in  $T.table$  into new-table
7      free  $T.table$ 
8       $T.table = new-table$ 
9       $T.size = 2 \cdot T.size$ 
10 insert  $x$  into  $T.table$ 
11  $T.num = T.num + 1$ 
```

لاحظ أن لدينا إجرائي "إدراج" هنا: إجراء TABLE-INSERT نفسه والإدراج الأساسي *elementary insertion* في جدول، في السطرين 10 و 11. يمكن أن نحلّل زمن تنفيذ TABLE-INSERT بدلالة عدد مرات الإدراج الأساسية بإسناد كلفة قيمتها 1 إلى كل عملية إدراج أساسية. نفترض أن زمن التنفيذ الفعلي لـ TABLE-INSERT خطّي بدلالة زمن إدراج العناصر الفردية، وبهذا تكون الكلفة المضافة اللازمة لتخصيص الجدول الابتدائي في السطر 2 ثابتة، بحيث تفوق كلفة نقل العناصر في السطر 6 الكلفة المضافة اللازمة لتخصيص منطقة التخزين وتخزينها في السطرين 5 و 7. نسّي الحدث الذي تنفّذ عنده السطور 9-5 توسيعًا *expansion*.



لنحلّل الآن متتالية من  $n$  عملية TABLE-INSERT مطبقة على جدول خيالي بدايةً. ما هي كلفة العملية  $c_i$  ذات الرقم  $i$ ؟ إذا كان ما يزال هناك مكان في الجدول الحالي (أو إذا كانت هذه هي العملية الأولى)، فإن  $i$  إذاً إنّنا بحاجة إلى تنفيذ عملية إدراج أساسية واحدة فقط في السطر 10. أما إذا كان الجدول الحالي ممتلئاً، وحدث توسيع، فإن  $c_i = i$ : كلفة تساوي 1 للإدراج الأساسي في السطر 10 يُضاف إليها  $i - 1$  لنسخ العناصر من الجدول القديم إلى الجدول الجديد في السطر 6. إذا نُقِذَت  $n$  عملية، فإن كلفة عملية في أسوأ الأحوال هي  $O(n)$ ، وهذا ما يؤدي إلى حدّ أعلى من المرتبة  $O(n^2)$  لزمن التنفيذ الكلي لـ  $n$  عملية TABLE-INSERT.

إن هذا الحد ليس محكماً كفاية، لأننا قلّما نوسّع الجدول خلال  $n$  عملية إدراج. تتسبب العملية  $i$  تحديداً في إحداث توسيع فقط عندما يكون  $i - 1$  من قوى 2 الصحيحة. إن الكلفة الممتدة لعملية ما هي في الحقيقة  $O(1)$ ، كما سنرى باستخدام التحليل المتّجّع. إن كلفة العملية ذات الرقم  $i$  هي

$$c_i = \begin{cases} i & \text{if } i - 1 \text{ is an exact power of 2} \\ 1 & \text{otherwise} \end{cases}.$$

إذن، الكلفة الكلية لـ  $n$  عملية TABLE-INSERT هي

$$\begin{aligned} \sum_{i=1}^n c_i &\leq n + \sum_{j=0}^{\lceil \lg n \rceil} 2^j \\ &< n + 2n \\ &= 3n, \end{aligned}$$

وذلك، لأن هناك  $n$  عملية على الأكثر كلفة كلٍّ منها 1، وكلف باقي العمليات تشكل سلسلة هندسية. ولما كانت الكلفة الكلية لـ  $n$  عملية TABLE-INSERT هي  $3n$ ، كانت الكلفة الممتدة للعملية الواحدة هي على الأكثر 3.

يمكننا، باستخدام طريقة المحاسبة، أن نشعر بوضوح أكبر لماذا يجب أن تكون الكلفة الممتدة لعملية TABLE-INSERT 3. إن كل عنصر يسدّد كلفة 3 عمليات إدراج أساسية: إدراج العنصر نفسه في الجدول الحالي، ونقل نفسه عند توسيع الجدول، وتحريك عنصر آخر كان قد انتقل مرة من قبل عندما توسّع الجدول. افترض على سبيل المثال، أن حجم الجدول هو  $m$  بعد عملية التوسيع مباشرة. إذن عدد العناصر في الجدول هو  $m/2$ ، ولا يمتلك الجدول أي رصيد. نحمل كل عملية إدراج 3 دولارات. يكلف الإدراج الأساسي الذي يحدث فوراً دولاراً واحداً. ونضع دولاراً آخر في رصيد العنصر للمضاف. ونضيف الدولار الثالث إلى رصيد أحد العناصر التي عددها  $m/2$  الموجودة في الجدول. لن يمتلك الجدول ثانية حتى نضيف  $m/2 - 1$  عنصراً إضافياً، وهكذا عندما يحوي الجدول  $m$  عنصراً ويصبح ممتلئاً، نكون قد وضعنا في رصيد كل عنصر دولاراً يدفعه ليعيد إدراج نفسه أثناء عملية التوسيع.

يمكن أيضًا استخدام طريقة الكمون لتحليل متتالية من  $n$  عملية TABLE-INSERT، ويجب أن نستخدمها في المقطع 4.17-2 لنصمم عملية TABLE-DELETE كلفتها للمخمدة هي أيضًا  $O(1)$ . نبدأ بتعريف دالة كمون  $\Phi$  تكون معدومة بعد عملية التوسيع مباشرة، إلا أنها تزداد حتى تصبح قيمتها مساوية لطول الجدول في اللحظة التي يصبح فيها الجدول ملاً، وهذا يمكننا التسديد لعملية التوسيع التالية باستخدام هذا الكمون. إن الدالة

$$\Phi(T) = 2 \cdot T.num - T.size \quad (5.17)$$

هي أحد الخيارات. ويكون لدينا  $T.num = T.size/2$ ، بعد عملية التوسيع مباشرة. وهكذا يكون  $\Phi(T) = 0$  مثلما نرغب. أما قبل عملية التوسيع مباشرة، فيكون  $T.num = T.size$ ، وهكذا يكون  $\Phi(T) = T.num$ ، مثلما نرغب. إن القيمة البدائية للكمون هي 0، ولما كان الجدول على الدوام نصف ملاً على الأقل، فإن  $T.num \geq T.size/2$  وهذا يقتضي أن تكون الدالة  $\Phi(T)$  موجبة دوماً. وهكذا فإن مجموع الكلف المخمدة لـ  $n$  عملية TABLE-INSERT يمثل حداً أعلى لمجموع الكلف الفعلية.

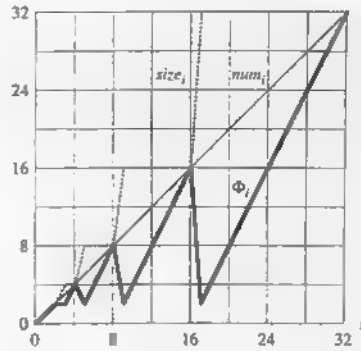
لتحليل الكلفة المخمدة لعملية TABLE-INSERT ذات الرقم  $i$ ، نحمل  $num_i$  يمثل عدّة العناصر المخزنة في الجدول بعد العملية ذات الرقم  $i$ ، و  $size_i$  يمثل الحجم الكلي للجدول بعد العملية ذات الرقم  $i$ ، و  $\Phi_i$  الكمون بعد العملية ذات الرقم  $i$ . بدايةً، يكون لدينا  $num_0 = 0$  و  $size_0 = 0$  و  $\Phi_0 = 0$ . إذا لم تسبب العملية ذات الرقم  $i$  في توسيع ما، يكون لدينا  $size_i = size_{i-1}$ ، وتكون الكلفة للمخمدة لهذه العملية:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= 1 + (2 \cdot num_i - size_i) - (2(num_i - 1) - size_i) \\ &= 3. \end{aligned}$$

أما إذا تسببت العملية ذات الرقم  $i$  في توسيع ما، فيكون عندها  $size_i = 2 \cdot size_{i-1}$  و  $size_{i-1} = num_{i-1} = num_i - 1$  وهذا يقتضي أن  $size_i = 2 \cdot (num_i - 1)$ . إذن الكلفة المخمدة للعملية هي:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= num_i + (2 \cdot num_i - size_i) - (2 \cdot num_{i-1} - size_{i-1}) \\ &= num_i + (2 \cdot num_i - 2 \cdot (num_i - 1)) - (2(num_i - 1) - (num_i - 1)) \\ &= num_i + 0 - (num_i - 1) \\ &= 1. \end{aligned}$$

يؤنّ الشكل 3.17 قيمة  $num_i$  و  $size_i$  و  $\Phi_i$  بدلالة  $i$ . لاحظ كيف أن الكمون يتزايد ليسدّد توسيع الجدول.



**الشكل 3.17** أثر متتالية من  $n$  عملية TABLE-INSERT على عدد العناصر  $num_i$  في الجدول، وعلى عدد الشقوق في الجدول  $size_i$ ، وعلى الكمون  $\Phi_i = 2 \cdot num_i - size_i$ ، حيث تقاس كل هذه المقادير بعد العملية ذات الرقم  $i$ . يبين الخط الرفيع  $num_i$ ، والخط المنقطع  $size_i$ ، والخط التخين  $\Phi_i$ . لاحظ أنه قبل عملية التوسيع مباشرة، يزداد الكمون حتى تبلغ قيمته عدد العناصر في الجدول، وبهذا يمكنه أن يسدّد لنقل كلِّ العناصر إلى الجدول الجديد. بعد ذلك، يهبط الكمون إلى 0، إلا أنه يرتفع مباشرة إلى 2 عند إدراج العنصر الذي تسبّب في التوسيع.

#### 2.4.17 توسيع الجدول وتقليصه

إن حذف عناصر محدّدة من الجدول بهدف تنحيز عملية TABLE-DELETE بسيط جدًّا. غير أننا كثيرًا ما نرغب في **تقليص** الجدول عندما يصبح عامل تحميل الجدول صغيرًا جدًّا، وذلك للحدّ من حجم الذاكرة المهدور. بمثلل تقليص الجدول توسيعه: عندما ينخفض عدد العناصر كثيرًا، فإننا نحصص جدولاً جديدًا أصغر منه، ثم ننسخ العناصر من الجدول القديم في الجدول الجديد. ويمكن عندها تحرير حيّز التخزين للجدول القديم بإعادته إلى نظام إدارة الذاكرة. نريد في الوضع المثالي أن نحافظ على الخاصيتين التاليتين:

- عامل تحميل الجدول محدود من الأسفل بثابت.
- والكلفة للمخمدة لعمليات الجدول محدودة من الأعلى بثابت.

نفترض أننا نقيس الكلفة بدلالة عمليات إدراج وحذف أساسية.

قد نتحدّ أن علينا مضاعفة حجم الجدول عند إضافة عنصر إلى جدول ملآن، وتقليص حجمه إلى النصف عندما تسبب عملية حذف في جعل الجدول أقل من نصف ملآن. نضمن هذه الاستراتيجية ألا ينخفض عامل التحميل أبدًا إلى أقل من  $1/2$ ، ولكنها لسوء الحظ قد تسبب في جعل الكلفة المخمدة لعملية ما كبيرًا فعلًا. لنأخذ السيناريو التالي: نقوم بـ  $n$  عملية على جدول  $T$ ، حيث  $n$  هي قوة صحيحة لـ 2. إن نصف العمليات الأولى هي عمليات إدراج، وهي تكلف وفق تحليلنا السابق كلفةً كلية  $\Theta(n)$ ، في نهاية متتالية

الإدراج، يكون  $T.num = T.size = n/2$ . وفيما يخص النصف الثاني من العمليات، فإننا ننفذ للمتتالية التالية:

insert, delete, delete, insert, insert, delete, delete, insert, insert, ....

تتسبب عملية الإدراج الأولى في توسيع الجدول إلى الحجم  $n$ . وتتسبب عمليتا الحذف التاليتان في تقليص الجدول من جديد إلى الحجم  $n/2$ . تتسبب عمليتا إدراج جديدتان في توسيع آخر، وهكذا. كلفة كل عملية توسيع وتقليص هي  $\Theta(n)$ ، وهناك  $\Theta(n)$  عملية منهما. وهكذا فإن الكلفة الكلية لـ  $n$  عملية هي  $\Theta(n^2)$ ، ومن ثم، فإن الكلفة للمجموعة لعملية واحدة هي  $\Theta(n)$ .

إن سبب هذه الاستراتيجية واضحة: بعد عملية توسيع، لا نقوم بعدد كافٍ من عمليات الحذف لنسدد للتقليص. وبالمثل، بعد التقليص لا نقوم بعدد كافٍ من عمليات الإدراج لنسدد للتوسيع.

يمكننا تحسين هذه الاستراتيجية بالسماح لعامل التحميل بالهبوط إلى أقل من  $1/2$ . وعلى وجه الخصوص، نتابع مضاعفة حجم الجدول عند إدراج عنصر في جدول ملآن، إلا أننا نقسم حجم الجدول على 2 عندما تتسبب عملية حذف في جعل الجدول ملآنًا إلى أقل من ربعه، بدلاً من نصفه. وبهذا يكون عامل التحميل محدودًا من الأسفل بالثابت  $1/4$ .

قد نعتبر بالجدول أن عامل تحميل يساوي  $1/2$  مثاليًا، ويكون عندها كمون الجدول 0. ومع ابتعاد عامل التحميل عن  $1/2$  بتزايد الكمون، وبذلك عندما يحين أوان توسيع الجدول أو تقليصه، يكون الجدول قد اكتسب ما يكفي من الكمون ليسدد نسخ كل العناصر في الجدول الجديد المخصص. إذن، نحتاج إلى دالة كمون تصل إلى  $T.num$  عندما يكون عامل التحميل قد ازداد إلى 1 أو تناقص إلى  $1/4$ . وبعد توسيع الجدول أو تقليصه، يعود عامل التحميل مجددًا إلى  $1/2$ ، ويهبط الكمون عائداً إلى 0.

لن ندرج هنا رماز TABLE-DELETE، إذ إنه مشابه لرماز TABLE-INSERT. وسنفترض لأغراض التحليل، أنه عندما ينخفض عدد العناصر في الجدول إلى 0، فإننا نحرر حيز خزن الجدول، أي إذا كان  $T.num = 0$  فإن  $T.size = 0$ .

بإمكاننا الآن استخدام طريقة الكمون لتحليل كلفة متتالية من  $n$  عملية TABLE-DELETE. نبدأ بتعريف دالة كمون  $\Phi$  تصبح 0 بعد التوسيع أو التقليص مباشرة، وتزداد بازدياد عامل التحميل إلى 1 أو تناقصه إلى  $1/4$ . نستقي عامل التحميل لجدول  $T$  غير خالٍ  $\alpha(T) = T.num/T.size$ . ولما كان  $T.num = T.size$  و  $\alpha(T) = 1$  للجدول الخالي، فلدينا دومًا  $T.num = \alpha(T) \cdot T.size$ ، سواءً أكان الجدول خاليًا أم لا. سنستخدم دالة الكمون

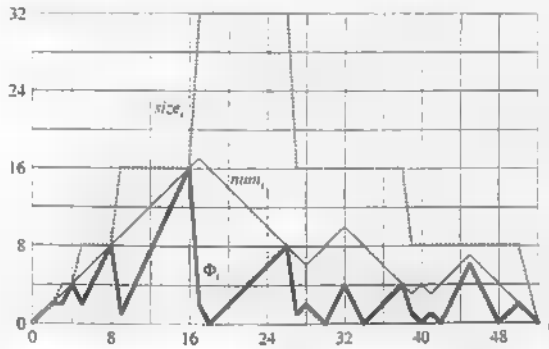
$$\Phi = \begin{cases} 2 \cdot T.num - T.size & \text{if } \alpha(T) \geq 1/2 \\ T.size/2 - T.num & \text{if } \alpha(T) < 1/2 \end{cases} \quad (6.17)$$

لاحظ أن كمون الجدول الخالي هو 0، وأن الكمون لا يصبح سالبًا أبدًا. إذن، فالكلفة للمجموعة الكلية لمتتالية

من عمليات اعتمادًا على  $\Phi$  هي حد أعلى للكلفة الفعلية لهذه المتتالية.

ستوقف قليلاً قبل البدء بالقيام بتحليل دقيق، لنتراقب بعض خواص دالة الكمون المبينة في الشكل 4.17. لاحظ أنه عندما يكون عامل التحميل  $1/2$ ، يكون الكمون  $0$ . وعندما يكون عامل التحميل  $1$ ، يكون لدينا  $T.size = T.num$ ، وهذا يقتضي أن  $\Phi(T) = T.num$ ، وبذلك يمكن للكمون أن يسدّد للتوسيع فيما إذا أُدرج عنصرٌ ما. عندما يكون عامل التحميل  $1/4$ ، يكون لدينا  $T.size = 4 \cdot T.num$ ، وهذا يقتضي أن  $\Phi(T) = T.num$ ، وبذلك يمكن للكمون أن يسدّد للتقليص فيما إذا حُلِفَ عنصرٌ ما.

لتحليل متتالية من  $n$  عملية  $TABLE-INSERT$  و  $TABLE-DELETE$ ، نفترض أن  $c_i$  الكلفة الفعلية للعملية ذات الرقم  $i$ ، و  $e_i$  الكلفة المختدة اعتمادًا على  $\Phi$ ، و  $num_i$  عدد العناصر المخزنة في الجدول بعد العملية ذات الرقم  $i$ ، و  $size_i$  حجم الجدول الكلي بعد العملية ذات الرقم  $i$ ، و  $\alpha_i$  عامل التحميل للجدول بعد العملية ذات الرقم  $i$ ، و  $\Phi_i$  الكمون بعد العملية ذات الرقم  $i$ . بدايةً، يكون  $num_0 = 0$  و  $size_0 = 0$  و  $\alpha_0 = 1$  و  $\Phi_0 = 0$ .



الشكل 4.17 أثر متتالية من  $n$  عملية  $TABLE-DELETE$  و  $TABLE-INSERT$  على  $num_i$  عدد العناصر في الجدول، و  $size_i$  عدد الشقوق في الجدول، والكمون  $\Phi_i$ .

$$\Phi_i = \begin{cases} 2 \cdot num_i - size_i & \text{if } \alpha_i \geq 1/2 \\ size_i / 2 - num_i & \text{if } \alpha_i < 1/2 \end{cases}$$

حيث يقاس كل من هذه القيم بعد العملية ذات الرقم  $i$ . يبيّن الخط الرفيع  $num_i$ ، والخط المتقطع  $size_i$ ، والخط النحيف  $\Phi_i$ ، لاحظ أنه قبل توسيع الجدول مباشرةً، يكون الكمون قد تنامي حتى أصبح يساوي عدد العناصر في الجدول، وبهذا يمكنه التصديد لنقل كل العناصر إلى الجدول الجديد. وبمثل، قبل التقليص مباشرةً، يكون الكمون قد تنامي حتى أصبح يساوي عدد العناصر في الجدول.

نبدأ بالحالة التي تكون فيها العملية ذات الرقم  $i$  هي TABLE-INSERT. إن التحليل الذي قمنا به مماثل للتحليل الذي عالجنا فيه توسيع الجدول في اللقطع 4.17-1 عندما  $\alpha_{i-1} \geq 1/2$ . وسواءً أتوسع الجدول أم لا، فإن كلفة العملية المخمدة  $\hat{c}_i$  هي 3 على الأكثر. وإذا كان  $\alpha_{i-1} < 1/2$ ، فلن يكون للجدول أن يتوسع. نتيجة لهذه العملية، إذ لا يمكن له التوسع إلا عندما  $\alpha_{i-1} = 1$ . وكذلك، إذا كان  $\alpha_i < 1/2$ ، فإن الكلفة المخمدة للعملية ذات الرقم  $i$  هي:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (\text{size}_i/2 - \text{num}_i) - (\text{size}_{i-1}/2 - \text{num}_{i-1}) \\ &= 1 + (\text{size}_i/2 - \text{num}_i) - (\text{size}_i/2 - (\text{num}_i - 1)) \\ &= 0.\end{aligned}$$

إذا كان  $\alpha_{i-1} < 1/2$ ، ولكن  $\alpha_i \geq 1/2$ ، فإن

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2 \cdot \text{num}_i - \text{size}_i) - (\text{size}_{i-1}/2 - \text{num}_{i-1}) \\ &= 1 + (2(\text{num}_{i-1} + 1) - \text{size}_{i-1}) - (\text{size}_{i-1}/2 - \text{num}_{i-1}) \\ &= 3 \cdot \text{num}_{i-1} - \frac{3}{2} \text{size}_{i-1} + 3 \\ &= 3\alpha_{i-1} \text{size}_{i-1} - \frac{3}{2} \text{size}_{i-1} + 3 \\ &< \frac{3}{2} \text{size}_{i-1} - \frac{3}{2} \text{size}_{i-1} + 3 \\ &= 3.\end{aligned}$$

وهكذا، فإن الكلفة للمخمدة لعملية TABLE-INSERT هي 3 على الأكثر.

تلفت الآن إلى الحالة التي تكون فيها العملية ذات الرقم  $i$  هي TABLE-DELETE. في هذه الحالة، يكون  $\text{num}_i = \text{num}_{i-1} - 1$ . إذا كان  $\alpha_{i-1} < 1/2$ ، وجب أن ندرس تسيب العملية في تقليص ما. فإذا لم تكن هذه هي الحالة، فإن  $\text{size}_i = \text{size}_{i-1}$ ، وتكون الكلفة للمخمدة للعملية:

$$\begin{aligned}\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (\text{size}_i/2 - \text{num}_i) - (\text{size}_{i-1}/2 - \text{num}_{i-1}) \\ &= 1 + (\text{size}_i/2 - \text{num}_i) - (\text{size}_i/2 - (\text{num}_i + 1)) \\ &= 2.\end{aligned}$$

وإذا كان  $\alpha_{i-1} < 1/2$  وتسيبت العملية ذات الرقم  $i$  في تقليص الجدول، فإن الكلفة الفعلية للعملية هي  $c_i = \text{num}_i + 1$ ، لأننا نحذف عنصرًا واحدًا وننقل  $\text{num}_i$  عنصرًا. ويكون لدينا  $\text{size}_i/2 = \text{size}_{i-1}/4$ ، والكلفة المخمدة للعملية هي:

$$\begin{aligned}
\hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\
&= (\text{num}_i + 1) + (\text{size}_i/2 - \text{num}_i) - (\text{size}_{i-1}/2 - \text{num}_{i-1}) \\
&= (\text{num}_i + 1) + ((\text{num}_i + 1) - \text{num}_i) - ((2 \cdot \text{num}_i + 2) - (\text{num}_i + 1)) \\
&= 1.
\end{aligned}$$

وعندما تكون العملية ذات الرقم  $i$  هي TABLE-DELETE، و  $\alpha_{i-1} \geq 1/2$ ، فإن الكلفة المختدة محدودة أيضاً من الأعلى بثابت، والتحليل متروك للتمرين 2-4.17. والخلاصة هي أنه، لما كانت الكلفة المختدة لكل عملية محدودة من الأعلى بثابت، فإن الزمن الفعلي لأية متتالية من  $\blacksquare$  عملية على جدول ديناميكي هي  $O(n)$ .

### تعاريف

#### 1-4.17

افترض أننا نريد تنجيز جدول تليد ديناميكي مفتوح العناصر. لماذا ينبغي أن نعتبر أن الجدول ملائ عندما يصل عامل تحميله إلى قيمة ما  $\alpha$  أصغر تماماً من 1؟ صِفْ بإيجاز كيف تجعل الإدراج في جدول تليد ديناميكي مفتوح العناصر يُنفَّذ بطريقة تجعل توقع الكلفة المختدة لعملية الإدراج هي  $O(1)$ . لماذا لا يكون توقع الكلفة الفعلية لعملية الإدراج بالضرورة  $O(1)$  لكل عمليات الإدراج؟

#### 2-4.17

بيِّنْ أنه إذا كان  $\alpha_{i-1} \geq 1/2$  وكانت العملية ذات الرقم  $i$  على الجدول الديناميكي هي TABLE-DELETE، فإن الكلفة المختدة للعملية اعتماداً على دالة الكمون في (6.17) محدودة من الأعلى بثابت.

#### 3-4.17

افترض أننا بدلاً من أن نقُلص جدولاً ما بتقسيم حجمه على 2 عندما يهبط عامل تحميله إلى أقل من  $1/4$ ، نقُلصه بضرب حجمه بـ  $2/3$  عندما يهبط عامل تحميله إلى  $1/3$ . وذلك باستخدام دالة الكمون

$$\Phi(T) = |2 \cdot T.\text{num} - T.\text{size}|,$$

بيِّنْ أن الكلفة المختدة لعملية TABLE-DELETE التي تعتمد هذه الاستراتيجية محدودة من الأعلى بثابت.

## مسائل

### 1-17 عداد اثنائي معكوس البتات

يدرس الفصل 30 خوارزمية هامة اسمها تحويل فورييه السريع، أو FFT. تقوم أول خطوة من الخوارزمية FFT بتنفيذ تبديل يعكس البتات *bit-reversal permutation* على جدول دخل  $A[0..n-1]$ ، حيث

$n = 2^k$ ، حيث  $k$  عدد صحيح غير سالب. يقوم هذا التبديل بمبادلة كل عنصرين من الجدول تمثيل دليل أحدهما الاثنائي عكس تمثيل دليل الثاني الاثنائي.

يمكننا أن نعبّر عن كل دليل  $k$  بتاً  $(a_{k-1}, a_{k-2}, \dots, a_0)$ ، حيث  $a = \sum_{i=0}^{k-1} a_i 2^i$ . نعرف

$$\text{rev}_k((a_{k-1}, a_{k-2}, \dots, a_0)) = (a_0, a_1, \dots, a_{k-1}) ;$$

إذن

$$\text{rev}_k(a) = \sum_{i=0}^{k-1} a_{k-i-1} 2^i .$$

فمثلاً، إذا كان  $n = 16$  (أو مكافئه  $k = 4$ )، فإن  $\text{rev}_k(3) = 12$ ، وذلك لأن التمثيل ذي الأربع بتات لـ 3 هو 0011، والذي يصبح عند عكسه 1100، وهو التمثيل ذي الأربع بتات لـ 12.

أ. لتكن  $\text{rev}_k$  دالة تُنفَّذ في زمن  $\Theta(k)$ ، اكتب خوارزمية لتنفيذ التبديل الذي يعكس البتات لصيغة طولها  $n = 2^k$  في زمن  $O(nk)$ .

يمكننا استخدام خوارزمية تعتمد على التحليل للمحدد لتحسّن زمن تنفيذ التبديل الذي يعكس البتات. نحافظ على "عداد معكوس البتات" وعلى إجراء BIT-REVERSED-INCREMENT الذي يعطي عند إعطائه قيمة  $a$  للعداد للمعكوس البتات، القيمة  $\text{rev}_k(\text{rev}_k(a) + 1)$ . فإذا كان، مثلاً  $k = 4$  والعداد للمعكوس البتات يبدأ عند الصفر، فإن الاستدعاءات للتالية لـ BIT-REVERSED-INCREMENT تولّد للتالية:

$$0000, 1000, 0100, 1100, 0010, 1010, \dots = 0, 8, 4, 12, 2, 10, \dots$$

ب. افترض أن الكلمات في حاسوبك تخزّن قيمًا من  $k$  بتاً، وأن بمقدور حاسوبك معالجة القيم الاثنائية في واحدة الزمن بتطبيق عمليات مثل الانزياح يسارًا أو يمينًا بمقادير اعتباطية، والمطف على مستوى البت bitwise-AND، والفصل على مستوى البت bitwise-OR، إلخ. صيغ تنحيّرًا لإجراء BIT-REVERSED-INCREMENT يسمح بتنفيذ تبديل يعكس البتات على جدول من  $n$  عنصرًا بزمن كلي  $O(n)$ .

ت. افترض أنه يمكنك إزاحة كلمة يسارًا أو يمينًا بتاً واحدًا فقط في واحدة الزمن. هل مازال ممكناً تنحيّر تبديل يعكس البتات في زمن  $O(n)$ .

## 2-17 جعل البحث الثنائي ديناميكيًا

يستغرق البحث الثنائي لصيغة مفروزة زمنًا لغزيميًا للقيام بالبحث، إلا أن زمن إدراج عنصر جديد خطي بدلالة حجم الجدول. يمكننا تحسين زمن الإدراج بالاحتفاظ بعدّة جداول مفروزة.

لنفترض، بوجه خاص، أننا نرغب في دعم SEARCH و INSERT على مجموعة من  $n$  عنصرًا. وليكن



$k = \lceil \lg(n+1) \rceil$ ، وليكن التمثيل الثنائي  $J = \{n_{k-1}, n_{k-2}, \dots, n_0\}$ . لدينا  $k$  صفيقة مفروزة  $A_0, A_1, \dots, A_{k-1}$ ، حيث طول الصفيقة  $A_i$  هو  $2^i$  لكل  $i = 0, 1, \dots, k-1$ . إن كل صفيقة ستكون مملأة أو فارغة، تبعاً لكون  $n_i = 1$  أو  $n_i = 0$  على الترتيب. وبهذا يكون عدد العناصر المخزنة في الصفيقات، التي عددها  $k$ ، هو  $\sum_{i=0}^{k-1} n_i 2^i = n$ . ومع أن كل صفيقة مستقلة مفروزة بحُد ذاتها، فلا توجد علاقة محددة بين عناصر صفيقات مختلفة.

أ. صيغ كيف يمكن أداء عملية SEARCH في بنية للعطيات هذه. وحلّل زمن تنفيذها في أسوأ الحالات.

ب. صيغ كيف يمكن إدراج عنصر جديد في بنية للعطيات هذه. وحلّل زمن تنفيذ هذه العملية في أسوأ الحالات، وزمن تنفيذها للمحدّد.

ت. ناقش كيف يمكن تنجز DELETE.

### 3-17 الأشجار ذات الثقل المتوازن المعتمدة

لنأخذ شجرة بحث ثنائية عادية، ونُفِّها بأن نضيف إلى كل عقدة  $x$  الوصفة  $x.size$  الذي يعطي عدد المفاتيح المخزنة في الشجرة الفرعية التي جذرها  $x$ . ليكن  $\alpha$  ثابتاً في المجال  $1/2 \leq \alpha < 1$ . نقول عن عقدة معطاة  $x$  إنها  $\alpha$ -متوازنة  $\alpha$ -balanced إذا كان  $x.left.size \leq \alpha \cdot x.size$  و  $x.right.size \leq \alpha \cdot x.size$ . ونقول عن شجرة بأسرها أنها  $\alpha$ -متوازنة إذا كانت كل عقدة من عقدها  $\alpha$ -متوازنة. اقترح G. Varghese النهج للمحدّد التالي للمحافظة على أشجار متوازنة الثقل.

أ. إن شجرة  $1/2$ -متوازنة، بمعنى ما، هي شجرة متوازنة قدر الإمكان. إذا كان لديك عقدة  $x$  في شجرة بحث ثنائية اعتباطية، بيّن كيف يُمكن إعادة بناء الشجرة الفرعية التي جذرها  $x$  بحيث تصبح  $1/2$ -متوازنة. يجب أن تُنفَّذ خوارزمتك في زمن  $\Theta(x.size)$  ومقدورها استخدام  $O(x.size)$  مساحة تخزين إضافية مساعدة.

ب. بيّن أن إجراء بحث في شجرة بحث ثنائية  $\alpha$ -متوازنة من  $n$  عقدة، يستغرق في أسوأ الحالات زمناً  $O(\lg n)$ .

افترض فيما تبقى من هذه المسألة أن الثابت  $\alpha$  أكبر غاملاً من  $1/2$ . لنفترض أننا أنجزنا الإجراءين INSERT و DELETE بالطريقة المعتادة للتعامل مع شجرة بحث ثنائية من  $n$  عقدة، إلا أنّهما وبعد كل عملية، إذا فقدت إحدى العقد في الشجرة خاصة  $\alpha$ -متوازنة، فإنه يُعاد بناء الشجرة الفرعية التي جذرها أعلى عقدة غير متوازنة، كهذه العقدة، لتصبح  $1/2$ -متوازنة.

سوف نحلّل نفع إعادة البناء هذا باستخدام طريقة الكمون. لنكن  $x$  عقدة في شجرة بحث ثنائية  $T$ ،

نعرف

$$\Delta(x) = |x.left.size - x.right.size| ,$$

ونعرف كمون  $T$  على أنه

$$\Phi(T) = c \sum_{x \in T: \Delta(x) \geq 2} \Delta(x) ,$$

حيث  $c$  ثابت كبير كفاية ويتعلق بـ  $\alpha$ .

ت. ناقش أن كمون أبة شجرة بحث ثنائية غير سالب، وأن كمون الشجرة  $1/2$ -متوازنة معلوم.

ث. افترض أن  $m$  وحدة كمون كافية لتسديد إعادة بناء شجرة فرعية ذات  $m$  عقدة. كم يجب أن يكون كير  $c$  بدلالة  $\alpha$ ، حتى يستغرق إعادة بناء شجرة فرعية غير  $\alpha$ -متوازنة زمنًا  $O(1)$ ؟

ج. بين أن كلفة إدراج عقدة في شجرة ذات  $n$  عقدة  $\alpha$ -متوازنة أو حذفها منها هو  $O(\lg n)$ .

#### 4-17 كلفة إعادة تنظيم بنية أشجار حمراء-سوداء

هناك أربع عمليات أساسية على الأشجار الحمراء-سوداء تُجري تغييرات بنيوية *structural modifications*: إدراج العقدة، وحذف العقدة، والدورانات وتغيير اللون. رأينا سابقًا أن RB-INSERT و RB-DELETE يستخدمان  $O(1)$  دورانًا فقط، وإدراج عقدة وحذف عقدة للمحافظة على الخواص الحمراء-السوداء، إلا أنهما قد يقومان بالزهد من عمليات تغيير اللون.

أ. صِفْ شجرة حمراء-سوداء نظامية ذات  $n$  عقدة بحيث يتسبب استدعاء RB-INSERT في إدراج العقدة ذات الرقم  $n + 1$  فيها عمليات تغيير لون من الرتبة  $\Omega(\lg n)$ . ثم صِفْ شجرة حمراء-سوداء نظامية ذات  $n$  عقدة بحيث يتسبب استدعاء RB-DELETE في حذف عقدة محددة عمليات تغيير لون من الرتبة  $\Omega(\lg n)$ .

على الرغم من أن عدد عمليات تغيير اللون قد يكون لغايتهمًا للعملية الواحدة في أسوأ الحالات، إلا أننا سنبرهن أن أية متتالية من  $m$  عملية RB-INSERT و RB-DELETE على شجرة حمراء-سوداء خالية بذاتٍ تتسبب في  $O(m)$  تغييرًا بنيويًا في أسوأ الحالات. لاحظ أننا نعد كل تغيير لون تغييرًا بنيويًا.

ب. إن بعض الحالات التي تعالجها الحلقة الرئيسية في رماز كلٍّ من RB-INSERT-FIXUP و RB-DELETE-FIXUP هي حالات *إنهاء تنفيذ* *terminating* أي عندما تتحقق مثل هذه الحالة يتوقف تنفيذ الحلقة بعد عدد ثابتٍ من العمليات الإضافية. حدّد أيّ الحالات في كلٍّ من RB-INSERT-FIXUP و RB-DELETE-FIXUP هي حالات إنهاء تنفيذ، وأنها ليس حالات إنهاء. (تلميح: انظر إلى الأشكال 5.13 و 6.13 و 7.13.)

سندرس أولاً التغييرات البنيوية عند القيام بعمليات إدراج فقط. لنكن  $T$  شجرة حمراء-سوداء، ونعرف  $\Phi(T)$  على أنه عدد العقد الحمراء فيها. افترض أنه يمكن لوحدة كمون واحدة أن تسدّ للتغيرات البنيوية التي تحدث في أية حالة من حالات RE-INSERT-FIXUP الثلاث.

ث. لنكن  $T'$  الشجرة الناتجة من تطبيق الحالة 1 من RB-INSERT على  $T$ . ناقش أن

$$\Phi(T') = \Phi(T) - 1$$

ث. عندما ندرج عقدة في شجرة حمراء-سوداء باستخدام RB-INSERT، يمكننا تجزئ العملية إلى ثلاثة أجزاء. اسرد التغييرات البنيوية والتغيرات الكمونية الناتجة عن الأسطر 1-16 من RB-INSERT، للحالات التي ليست حالات إنهاء في RB-INSERT-FIXUP، وأخيراً لحالات الإنهاء في RB-INSERT-FIXUP.

ج. ناقش، اعتماداً على الجزء (ث) أن عدد التغييرات البنيوية المختدة المنفذة عند أي استدعاء ل RB-INSERT هي  $O(1)$ .

نرغب الآن في أن نرهن أن هناك  $O(m)$  تغييراً بنيوياً عندما توجد عمليات إدراج وحذف. نعرف لكل عقدة  $x$ :

$$w(x) = \begin{cases} 0 & \text{إذا كانت } x \text{ حمراء} \\ 1 & \text{إذا كانت } x \text{ سوداء وليس لها أبناء حمراء} \\ 0 & \text{إذا كانت } x \text{ سوداء ولها ابن واحد أحمر} \\ 2 & \text{إذا كانت } x \text{ سوداء ولها ابنان أحمران} \end{cases}$$

نعيد الآن تعريف كمون شجرة حمراء-سوداء  $T$  على أنه

$$\Phi(T) = \sum_{x \in T} w(x) ,$$

ولنكن  $T'$  الشجرة الناتجة عن تطبيق أية حالة ماعدا حالات الإنهاء في RB-INSERT-FIXUP أو RB-DELETE-FIXUP على  $T$ .

ح. بين أن  $\Phi(T') \leq \Phi(T) - 1$  في كل الحالات التي لا يحدث فيها إنهاء في RB-INSERT-FIXUP. ناقش أن عدد التغييرات البنيوية المختدة للنفذة عند أي استدعاء ل RB-INSERT-FIXUP هو  $O(1)$ .

خ. بين أن  $\Phi(T') \leq \Phi(T) - 1$  في كل الحالات التي لا يحدث فيها إنهاء في RB-DELETE-FIXUP. ناقش أن عدد التغييرات البنيوية المختدة للنفذة عند أي استدعاء ل RB-DELETE-FIXUP هو  $O(1)$ .

د. أكمل برهان أن أية متتالية من  $m$  عملية RB-INSERT و RB-DELETE تُنجز، في أسوأ الحالات،  $O(m)$  تغييراً بنيوياً.

### 17-5 التحليل التنافسي للقوائم الذاتية التنظيم باستخدام كسبية النقل-إلى-المقدمة

**القائمة الذاتية التنظيم** *self-organizing list* هي قائمة مترابطة من  $n$  عنصرًا، ولكلٍّ من هذه العناصر مفتاحٌ وحيد. عندما نبحث عن عنصر في القائمة، نُعطى مفتاحًا، ونحاول العثور على العنصر الذي له هذا المفتاح.

للقائمة الذاتية التنظيم خاصيتان هامتان:

1. للعثور على عنصر في القائمة اعتمادًا على مفتاحه للمطابق، علينا عبور القائمة من البداية وحتى نصادف العنصر ذا المفتاح المطابق. فإذا كان هذا العنصر في المرتبة  $k$  من بداية القائمة، فإن كلفة العثور عليه هي  $k$ .
2. قد نعيد ترتيب عناصر القائمة بعد أية عملية، وفقًا لقاعدة معطاة بكلفة محددة. وقد نختار أية كسبية نريد لنقرر كيف نعيد ترتيب القائمة.

افترض أننا نبدأ بقائمة معطاة من  $n$  عنصرًا، وأتينا أعطينا متتالية نفاذ مؤلفة من المفاتيح  $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_m)$  للبحث عنها بالترتيب. كلفة المتتالية هي مجموع كلف النفاذ إلى كل مفتاح منها على حدة.

تتركز هذه المسألة، من بين كل الطرق المختلفة للمكينة لإعادة ترتيب القائمة بعد عملية ما، على المبادرة بين عناصر القائمة المتحاورة - أي تبديل مواقعها في القائمة - بكلفة قدرها واحد لكل عملية مبادلة. نبيّن فيما يلي - باستخدام دالة كمون - أن الكلفة الكلية لكسبية محددة لإعادة ترتيب القائمة، وهي كسبية النقل-إلى-المقدمة، لا تتجاوز أربع أضعاف أية كسبية أخرى يمكن استخدامها للمحافظة على ترتيب القائمة - ولو كانت الكسبية الأخرى على معرفة مسبقة بمتتالية النفاذ! نسقي هذا النوع من التحليل **بالتحليل التنافسي** *competitive analysis*.

نرمز إلى كلفة النفاذ إلى متتالية  $\sigma$  مع كسبية محددة  $H$  وترتيب بدلي للقائمة، بالرمز  $C_H(\sigma)$ . ليكن  $m$  عدد مرات النفاذ في  $\sigma$ .

أ. ناقش أنه إذا لم تكن الكسبية  $H$  تعرف متتالية النفاذ مسبقًا، فإن كلفة الحالة الأسوأ لـ  $H$  على متتالية نفاذ  $\sigma$  هي  $C_H(\sigma) = \Omega(mn)$ .

عند استخدام كسبية النقل-إلى-المقدمة *move-to-front*، بعد البحث عن عنصر  $x$  مباشرة، نقوم بنقل  $x$  إلى الموقع الأول في القائمة (أي مقدمة القائمة).

لنرمز بـ  $\text{rank}_L(x)$  إلى مرتبة العنصر  $x$  في القائمة  $L$ ، أي موقع  $x$  في القائمة  $L$ . فإذا كان  $x$ ، على سبيل المثال، العنصر الرابع في  $L$ ، فإن  $\text{rank}_L(x) = 4$ . ولنرمز بـ  $c_L$  إلى كلفة النفاذ إلى  $\sigma_L$  باستخدام كسبية

النقل-إلى-المقدمة، التي تتضمن كلفة إيجاد العنصر في القائمة وكلفة نقله إلى مقدمة القائمة باستخدام سلسلة من المبادلات بين عناصر القائمة المتجاورة.

ب. يبين أنه إذا كان  $\sigma_i$  يسمح بالنفاذ إلى العنصر  $x$  في القائمة  $L$  باستخدام كسبة النقل-إلى-المقدمة، فإن

$$c_i = 2 \cdot \text{rank}_L(x) - 1$$

نقارن الآن النقل-إلى-المقدمة بأية كسبة أخرى  $H$  تعالج متتالية نفاذ وفقاً للخاصتين المذكورتين في البداية. قد تقوم الكسبة  $H$  بمبادلة العناصر في القائمة بالطريقة التي ترتبها، وقد تكون على معرفة مسبقة بكامل متتالية النفاذ.

لتكن  $l_i$  القائمة بعد النفاذ إلى  $\sigma_i$  باستخدام النقل-إلى-المقدمة، ولتكن  $l_i^*$  القائمة بعد النفاذ إلى  $\sigma_i$  باستخدام الكسبة  $H$ . نرسم إلى كلفة النفاذ إلى  $\sigma_i$  باستخدام النقل-إلى-المقدمة بـ  $c_i$ ، وباستخدام الكسبة  $H$ ، بـ  $c_i^*$ . افترض أن الكسبة  $H$  تقوم بـ  $e_i^*$  مبادلة عند النفاذ إلى  $\sigma_i$ .

ث. لقد يثبت في الجزء (ب) أن  $c_i = 2 \cdot \text{rank}_{l_{i-1}}(x) - 1$ . يبين الآن أن

$$c_i^* = 2 \cdot \text{rank}_{l_{i-1}}(x) + e_i^*$$

نعرف الزوج المعكوس *inversion* في القائمة  $l_i$  على أنه زوج من العناصر  $y$  و  $z$  حيث يسبق العنصر  $y$  العنصر  $z$  في القائمة  $l_i$ ، فيما يسبق العنصر  $z$  العنصر  $y$  في القائمة  $l_i^*$ . افترض أن القائمة  $l_i$  فيها  $q_i$  زوجاً معكوساً بعد معالجة متتالية النفاذ  $(\sigma_1, \sigma_2, \dots, \sigma_i)$ . في هذه الحالة نعرف دالة كمون  $\Phi$  تقابل كل قائمة  $l_i$  بالعدد الحقيقي  $\Phi(l_i) = 2q_i$ . على سبيل المثال، إذا كانت القائمة  $l_i$  تضم العناصر  $(e, c, a, d, b)$ ، وكانت القائمة  $l_i^*$  تضم العناصر  $(c, a, b, d, e)$ ، فإن القائمة  $l_i$  تضم خمسة أزواج معكوسة  $((e, c), (e, a), (e, d), (e, b), (d, b))$ ، وهذا يكون  $\Phi(l_i) = 10$ . لاحظ أن  $\Phi(l_i) \geq 0$  مهما كانت  $i$ ، وأن كلاً من النقل-إلى-المقدمة والكسبة  $H$  تبدآن مع القائمة نفسها  $l_0$ ، إذن  $\Phi(l_0) = 0$ .

ث. ناقش أن أية مبادلة إما أن تزيد الكمون بمقدار 2 أو تُنقصه بـ 2.

افترض أن النفاذ  $\sigma_i$  يعثر على العنصر  $x$ . ولكي نعرف كيف يتغير الكمون بسبب  $\sigma_i$ ، سنجزئ العناصر ما عدا  $x$  إلى 4 مجموعات، اعتماداً على مواقعها في القوائم تماماً قبل النفاذ ذي الترتيب  $i$ :

- المجموعة  $A$ ، تتألف من العناصر التي تسبق  $x$  في كلٍّ من  $l_{i-1}$  و  $l_i^*$ .
- المجموعة  $B$ ، تتألف من العناصر التي تسبق  $x$  في  $l_{i-1}$  وتنبع في  $l_i^*$ .
- المجموعة  $C$ ، تتألف من العناصر التي تنبع  $x$  في  $l_{i-1}$  وتسبقه في  $l_i^*$ .
- المجموعة  $D$ ، تتألف من العناصر التي تنبع  $x$  في كلٍّ من  $l_{i-1}$  و  $l_i^*$ .

ج. ناقش أن  $\text{rank}_{L_{i-1}}(x) = |A| + |B| + 1$  وأن  $\text{rank}_{L_{i-2}}(x) = |A| + |C| + 1$ .

ح. بَيِّن أن النفاذ  $\sigma_i$  يغير الكمون بالمقدار

$$\Phi(L_i) - \Phi(L_{i-2}) \leq 2(|A| - |B| + t_i')$$

حيث، كما ذكرنا سابقًا، تقوم الكسبية  $H$  بـ  $t_i^*$  مبادلة خلال النفاذ  $\sigma_i$ .

عرّف كلفة النفاذ  $\sigma_i$  المخمّدة  $\varepsilon_i$  بـ  $\varepsilon_i = c_i + \Phi(L_i) - \Phi(L_{i-1})$ .

خ. بَيِّن أن كلفة النفاذ  $\sigma_i$  المخمّدة  $\varepsilon_i$  محدودة من الأعلى بالمقدار  $4c_i'$ .

د. استنتج أن  $C_{MTP}(\sigma)$  كلفة النفاذ إلى مفاتيح للتالية  $\sigma$  باستخدام النقل-إلى-المقدمة هي على الأكثر

أربعة أضعاف  $C_H(\sigma)$  كلفة النفاذ إلى مفاتيح للتالية  $\sigma$  باستخدام أية كسبية  $H$ ، بفرض أن كلتا الكسبيتين تبدأان مع القائمة نفسها.

## ملاحظات الفصل

استخدم Aho و Hopcroft و Ullman [5] التحليل المجمع لتحديد زمن تنفيذ عمليات على غابة من المجموعات المنفصلة disjoint-set؛ سنحلل بنية المعطيات هذه باستخدام طريقة الكمون في الفصل 21. يقوم Tarjan [331] بمسح لطريقي المحاسبة والكمون في التحليل للمخمد، ويقدم تطبيقات عدّة. وهو ينسب طريقة المحاسبة إلى عدّة مؤلفين، منهم M.R. Brown و R.E. Tarjan و S. Huddleston و K. Mehlhorn. وينسب طريقة الكمون إلى D.D. Sleator. ويعود مصطلح "amortized" مخمد إلى D.D. Sleator و R.E. Tarjan.

إن دوال الكمون ذات فائدة أيضًا في برهان حدود دنيا في بعض أغاط المسائل. نعرّف لكل تشكيلة من المسألة، دالة كمون تقابل بين التشكيلة وعدد حقيقي. ثم نحدّد كمون التشكيلة البدائية  $\Phi_{\text{init}}$ ، وكمون التشكيلة النهائية  $\Phi_{\text{final}}$ ، والتغير الأعظم في الكمون  $\Delta\Phi_{\text{max}}$  الناتج عن أية خطوة، وعليه لا بد أن يكون عدد الخطوات على الأقل  $|\Phi_{\text{final}} - \Phi_{\text{init}}| / \Delta\Phi_{\text{max}}$ . هناك أمثلة عن استخدام دوال الكمون في برهان حدود دنيا لتعقيد عمليات I/O في أعمال Cormen و Sundquist و Wisniewski [80]، و Floyd [107] و Aggarwal و Vitter [4]. طبق Krumme و Cybenko و Venkataraman [221]، دوال الكمون لبرهان حدود دنيا على البث الشامل gossiping: تبادل عنصر وحيد من كل عقدة في بيان إلى باقي العقد الأخرى.

تعمل كسبية النقل-إلى-المقدمة من المسألة 17-5 جيئًا في الحالة العملية. يضاف إلى ذلك أننا إذا أقررتنا

بأننا عندما نعثر على عنصر، فإننا نستخرجه من موقعه في القائمة ونضعه في مقدمتها في زمن ثابت،  
 فيإمكاننا أن نبرهن أن كلفة النقل-إلى-للقدمة هي على الأكثر ضعفي كلفة أية كسبية أخرى، ومن ضمنها،  
 مرة أخرى، الكسبية التي تُعرف مسبقًا كامل متتالية النفاذ.







## تمهيد

يعود هذا الباب إلى دراسة بني المخططات التي تدعم العمليات على المجموعات الديناميكية، ولكن على مستوى أكثر تقدماً من الباب III. فأتان من فصول هذا الباب، على سبيل المثال، يستخدمان تقنيات التحليل المتعدد - التي رأيناها في الفصل 17 - استخداماً موسعاً.

يقدم الفصل 18 الأشجار المعتمدة B-trees، وهي أشجار بحث متوازنة صُممت لتخزن على الأقراص بوجه خاص. ولما كانت الأقراص تعمل على نحو أبطأ بكثير من الذواكر ذات النفاذ العشوائي (الذواكر الحية)، فإننا لا نقيس أداء الأشجار المعتمدة بمقدار زمن الحساب الذي تستغرقه العمليات على المجموعات الديناميكية فحسب، بل أيضاً بعدد عمليات النفاذ إلى القرص التي تؤديها. يزداد عدد مرات النفاذ إلى القرص في كل عملية مع ارتفاع الشجرة المعتمدة، لكن العمليات على الأشجار المعتمدة تحافظ على ارتفاع صغير لها.

يعطي الفصل 19 تيجراً للكومات القابلة للدمج mergeable heap، التي تدعم العمليات INSERT و MINIMUM و EXTRACT-MIN و UNION.<sup>1</sup> تؤخذ عملية UNION كومتين أو تدبجهما. وتدعم كومات فيوناتشي Fibonacci heaps - بنية للمخططات في الفصل 19 - أيضاً عمليتي DELETE و DECREASE-KEY. نستخدم حدوداً زمنية معتمدة لقياس أداء كومات فيوناتشي. تستغرق عمليات INSERT و MINIMUM و EXTRACT-MIN زمناً فعلياً ونحسباً  $O(1)$  فقط في كومات فيوناتشي، وتستغرق عمليتا EXTRACT-MIN و DELETE زمناً نحسباً  $O(\lg n)$ . لكن الفائدة الأكثر أهمية لكومات فيوناتشي هي أن DECREASE-KEY

<sup>1</sup> كما في المسألة 2-10، عرّفنا كومة قابلة للدمج لدعم MINIMUM و EXTRACT-MIN، وبذلك يمكننا أيضاً تسميتها كومات قابلة للدمج وفق الأصغر mergeable min-heap. بالمقابل، إذا كانت تدعم MAXIMUM و EXTRACT-MAX، فتكون كومات قابلة للدمج وفق الأكبر mergeable max-heap. نقصد بالكومات القابلة للدمج الكومات القابلة للدمج وفق الأصغر، ما لم نُشر إلى خلاف ذلك.

تستغرق زمنًا محددًا  $O(1)$  فقط. ولما كانت عملية DECREASE-KEY تستغرق زمنًا محددًا ثابتًا، فإنَّ كومات فيوناتشي هي مركبات أساسية في بعض أسرع الخوارزميات مقارنةً حاليًا في مسائل البيانات.

مع ملاحظة أننا يمكن أن نقلب على الحد الأدنى للفرز،  $\Omega(n \lg n)$ ، حين تكون المفاتيح أعدادًا صحيحة ضمن مجال محدد، يساءل الفصل 20 عن إمكان تصميم بنية معطيات تدعم عمليات المجموعات الديناميكية SEARCH و INSERT و DELETE و MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR بزمن  $O(\lg n)$  عندما تكون للمفاتيح أعدادًا صحيحة ضمن مجال محدد. وتشير الإجابة إلى أننا نستطيع ذلك، باستخدام بنية معطيات عودية معروفة باسم شجرة van Emde Boas. إذا كانت المفاتيح أعدادًا صحيحة فريدة مسبوقة من المجموعة  $\{0, 1, 2, \dots, u-1\}$  حيث  $u$  قوة تامة للعدد 2، فإنَّ أشجار van Emde Boas تدعم العمليات المذكورة بزمن  $O(\lg \lg u)$ .

أخيرًا، يقدم الفصل 21 بنية للمجموعات المنفصلة. لدينا فضاء من  $n$  عنصرًا مجزأة إلى مجموعات ديناميكية. في البداية، ينتمي كل عنصر إلى مجموعته الوحيدة العنصر. تؤخذ عملية UNION مجموعتين، ويحدد الاستعلام FIND-SET المجموعة الوحيدة التي تتضمن عنصرًا معينًا حاليًا. بتعثيل كل مجموعة بشجرة بسيطة ذات جذر نحصل على عمليات مدهشة السرعة: سلسلة من  $\alpha(n)$  عملية تُنفَّذ بزمن  $O(m \alpha(n))$ ، حيث  $\alpha(n)$  هي دالة بطيئة النمو جدًا -  $\alpha(n)$  هي على الأكثر 4 في أي تطبيق يمكن تحيله. التحليل المحدث الذي يثبت هذا الحد الزمني معقد جدًا بمقدار بساطة بنية المعطيات.

ليست المواضيع المقدمة في هذا الباب هي الأمثلة الوحيدة على بنى المعطيات "المتقدمة" في أي حال من الأحوال. فتمتد بنى أخرى للمعطيات المتقدمة تتضمن ما يلي:

- **الأشجار الديناميكية Dynamic trees**، أدخلها Sleator و Tarjan [319] وناقشها Tarjan [330]، وهي تحتفظ بغابة من الأشجار المنفصلة ذات الجذر. كلُّ وصلة في كلِّ شجرة لها تكلفة ذات قيمة حقيقية. تدعم الأشجار الديناميكية استعلامات العثور على الآباء، والجذور، وتكاليف الوصلات، والوصلة الأقل تكلفة في طريق بسيط من عقدة ما إلى الجذر. يمكن معالجة الأشجار بقطع الوصلات، وتحديث تكلفة جميع الوصلات ضمن طريق بسيط من عقدة ما إلى الجذر، ووصل جذر إلى شجرة أخرى، وتحويل عقدة ما إلى جذر في الشجرة التي تظهر فيها. تعطي إحدى تنحيزات الأشجار الديناميكية حدًا زمنيًا محددًا  $O(\lg n)$  لكل عملية؛ على حين يعطي تنحيز أكثر تعقيدًا حدًا زمنيًا  $O(\lg n)$  في أسوأ الحالات. تُستخدم الأشجار الديناميكية في بعض أسرع خوارزميات تدفق الشبكات مقارنةً.
- **أشجار سيلاي Splay trees**، طوّرها Sleator و Tarjan [320]، وناقشها أيضًا Tarjan [330]، وهي أحد أشكال أشجار البحث الثنائي التي تُنفَّذ عمليات البحث للمباراة بزمن محدد  $O(\lg n)$ . إن أحد تطبيقات أشجار سيلاي هو تيسيط الأشجار الديناميكية.

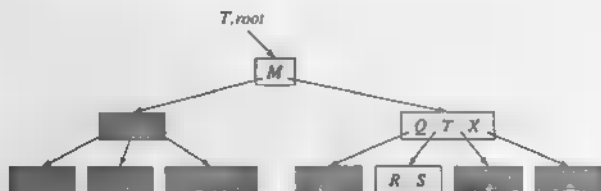
- **بنى المعطيات الدائمة Persistent**، تسمح بالاستعلامات إضافة إلى التحديثات في بعض الأحيان، وذلك على نسخ سابقة من بنى المعطيات. يقَدِّم Driscoll و Sarnak و Sleator و Tarjan [98] تقنيات لجعل بنية المعطيات المترابطة دائمة بزمَن وحجم قليلين. تعطي المسألة 13-1 مثلاً بسيطاً على مجموعة ديناميكية دائمة.
  - وكما في الفصل 20، تسمح عدة بنى معطيات بتحيز أسرع للعمليات الممحصية (DELETE و INSERT و SEARCH) في حالة فضاء محدود من المفاتيح. يمكن لهذه البنى، بالاستفادة من هذه التحديثات، الوصول إلى أزمَنة تنفيذ تقريبية، أفضل من بنى للمعطيات التي تعتمد على المقارنة، في أسوأ الحالات. أدخل Fredman و Willard **أشجار الصهر fusion trees** [115]، التي كانت أول بنية معطيات تسمح بعمليات ممحصية أسرع عندما يكون الفضاء مقتصرًا على الأعداد الصحيحة. وقد يَبْنَى كيفية تنجيز هذه العمليات بزمَن  $O(\lg n / \lg \lg n)$ . أعطت عدة بنى معطيات تالية، منها **أشجار البحث الأسية exponential search trees** [16] حدودًا محسَّنة على بعض العمليات الممحصية أو كلها، وهي مذكورة في ملاحظات الفصول ضمن الكتاب.
  - تدعم **بنى معطيات البيانات الديناميكية Dynamic graph data structures** استعلامات مختلفة حينما تسمح بتغيير بنية البيان باستخدام عمليات إدراج وحذف على العقد أو الوصلات. تتضمن أمثلة على الاستعلامات التي تدعمها: ترابط الرؤوس vertex connectivity [166]، وترابط الوصلات edge connectivity، وأشجار المسح الصغرى minimum spanning trees [165]، والترابط الثنائي bicconnectivity، والإغلاق المتعدي transitive closure [164].
- تُذكر ملاحظات الفصول في هذا الكتاب بنى معطيات إضافية.

الأشجار المعممة B-trees هي أشجار بحث متوازنة مصممة لتعمل جيدًا على الأقراص أو وسائط التخزين الثانوية الأخرى ذات النفاذ المباشر. تشبه الأشجار المعممة الأشجار الحمراء-السوداء red-black trees (الفصل 13)، لكنها تفضلها في تقليص عمليات الدخول والخروج I/O على القرص. نستخدم عددًا منظمًا قواعد معطيات الأشجار المعممة أو متغيرات منها، لحزن المعلومات.

تختلف الأشجار المعممة عن الأشجار الحمراء-السوداء في أن لعقدتها العديد من الأبناء، من بضعة أبناء إلى الآلاف. أي إن عامل التفرع "branching factor" للأشجار المعممة يمكن أن يكون ضخمًا جدًا، مع أن ذلك يتعلق عادةً بمميزات وحدة القرص المستخدمة. تشبه الأشجار المعممة الأشجار الحمراء-السوداء في أن لكل شجرة معمة من  $n$  عقدة ارتفاعًا هو  $O(\lg n)$ ، مع أن الارتفاع الدقيق لشجرة يمكن أن يكون أقل بكثير من ارتفاع شجرة حمراء-سوداء، وذلك لأن عامل التفرع فيها، ومن ثم قاعدة اللغاريتم التي تعبر عن الارتفاع، يمكن أن يكون أكبر بكثير. وبالتالي يمكننا استخدام الأشجار المعممة لتتجزئ عدة عمليات من عمليات المجموعات الديناميكية بزمن  $O(\lg n)$ .

إن أشجار B-trees هي التعميم الطبيعي لأشجار البحث الثنائية. يُظهر الشكل 1.18 شجرة معمة بسيطة. إذا تضمنت إحدى العقد الداخلية  $x$  في شجرة معمة  $x.n$  مفتاحًا، كان لـ  $x$  عددًا من الأبناء يساوي  $x.n + 1$ . يمكن الاستفادة من المفاتيح في العقدة  $x$  كنقاط تقسيم تفصل مجال المفاتيح الذي تعالجه  $x$  إلى  $x.n + 1$  مجالًا جزئيًا، يُعالج كلٌّ منها ابنًا من أبناء  $x$ . حين نبحث عن مفتاح في شجرة B-tree، فإننا نتخذ قرارًا بـ  $x.n + 1$  طريقة، اعتمادًا على  $x.n$  مفتاحًا مخزنًا في العقدة  $x$ . تختلف بنية العقد في الأوراق عن بنية العقد الداخلية؛ وسندرس هذه الاختلافات في المقطع 1.18.

يُعطي المقطع 1.18 تعريفًا دقيقًا لأشجار B-trees، ونبرهن على أن ارتفاع شجرة B-tree يزداد لغاريتميًا فقط مع عدد العقد التي تحتويها. ويصف المقطع 2.18 كيفية البحث عن مفتاح، وكيفية إدراج مفتاح في شجرة B-tree. ونباشق المقطع 3.18 عملية الحذف. ولكن، قبل أن نتابع، يجب أن نسال: لماذا نقيّم بنى المعطيات المصممة للعمل على الأقراص تقييماً مختلفًا عن بنى المعطيات للمصممة للعمل في الذاكرة الرئيسية العشوائية النفاذ.



**الشكل 1.18** شجرة B-tree، مفاتيحها هي الحروف الصوامت في اللغة الإنكليزية. لكل عقدة داخلية  $x.n + 1$  مفتاحاً  $x.n + 1$  ابنًا. كل الأوراق على نفس العمق من الشجرة. العقد المظلمة تظليلاً خفيفاً هي التي جرى فحصها للبحث عن الحرف  $R$ .

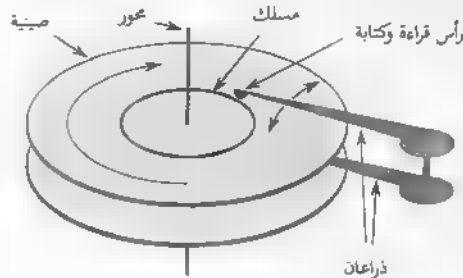
### بني المعطيات في الخزن الثانوي

يستفيد نظام حاسوبي من تقانات مختلفة عديدة توفرّ سعات في الذاكرة. تتألف **الذاكرة الأولية** *primary memory* (أو **الذاكرة الرئيسية** *main memory*) لنظام حاسوبي من رقائق ذاكرة سليكونية. إن هذه التقانة، عموماً، أكثر تكلفة لكل بت عزّز من تقانة الخزن للمغناطيسي كما في الأشرطة المغناطيسية والأقراص. تحتوي معظم النظم الحاسوبية أيضاً على **خزن ثانوي** *secondary storage* يعتمد على الأقراص المغناطيسية؛ يزيد مقدار هذا الخزن الثانوي غالباً عن مقدار الذاكرة الرئيسية بمئتين على الأقل.

يُظهر الشكل 2.18 سواقة أقراص *disk drive* غودجية. تتألف السواقة من صيّفة *platter* أو أكثر، تدور بسرعة ثابتة حول **محور** *spindle* مشترك. تغطي سطح كل صينية مادة مغناطيسية. تقرأ السواقة كلّ صينية وتكتب عليها بواسطة رأس *head* يقع في نهاية ذراع *arm*. يمكن أن تحرك الأذرع رؤوسها باتجاه المحور أو بعيداً عنه. حين يكون رأس ما مستقرّاً، يُسمى السطح الذي يمرّ تحته مباشرة **مسلكاً** *track*. إن تعدّد الصينيات يزيد في سعة سواقة القرص فقط، ولا يزيد في أدائها.

ومع أن الأقراص أقل تكلفة وأكبر سعة من الذاكرة الرئيسية، فإنها أبطأ بكثير لأن فيها أجزاء ميكانيكية متحركة.<sup>١</sup> تتألف الحركة الميكانيكية من مكونين: دوران الصينيات وحركة الذراع. في وقت كتابة هذا الكتاب، كانت الأقراص المتوفرة تدور بسرعات 5400–15000 دورة في الدقيقة (RPM). نرى عادةً سرعات 15000 RPM في السواقات على مستوى المخدمات، وسرعات 7200 RPM في سواقات الحواسيب الشخصية، وسرعات 5400 RPM في سواقات الحواسيب المحمولة. ومع أن 7200 RPM قد تبدو سرعة كبيرة، إلا أن كل دورة تأخذ 8.33 ميلي ثانية. وهذا تقريباً، أطول بخمس مرات من 50 نانو ثانية (أكثر أو

<sup>١</sup> أثناء كتابة هذا الكتاب وصلت سواقات صلبة الحالة *solid-state* إلى سوق المستهلك. ومع أنّ هذه السواقات أسرع من سواقات الأقراص الميكانيكية، إلا أنها تكلف أكثر لكل جيجابايت وسعاً أقل من سعات سواقات الأقراص الميكانيكية.



**الشكل 2.18** سواقة قرص غوذجية. تتألف من عدة صينيات تدور حول محور (تظهر هنا صينيتان). تجري القراءة من كل صينية أو الكتابة عليها بواسطة رأس في نهاية ذراع. الأذرع مجموعة معاً بحيث تحرك رؤوسها بانسجام. تدور الأذرع حول محور دوران مشترك. للمسلك هو السطح الذي يمر تحت رأس القراءة أو الكتابة حين يكون مستقرًا.

أقل، الذي هو زمن النفاذ الشائع إلى ذاكرة السيليكون. وبمعبر آخر، إذا كان علينا الانتظار دورة كاملة لتقع مادة ما تحت رأس القراءة أو الكتابة، أمكننا النفاذ إلى الذاكرة الرئيسية أكثر من 100,000 مرة خلال ذلك المسح. وسطًا يجب أن نتظر نصف دورة فقط، لكن يبقى الفرق بين زمن النفاذ إلى ذاكرة السيليكون مقارنة بزمن النفاذ إلى الأقراص كبيرًا. يأخذ تحريك الذراع أيضًا بعض الوقت. في وقت كتابة هذا النص، تقع أزمته النفاذ للأقراص المعهودة ضمن المجال 11 إلى 11 ميلي ثانية.

ولكي نحفض زمن انتظار الحركات الميكانيكية، نُلزم الأقراص بالنفاذ إلى عدة مواد في الوقت نفسه بدلاً من أن تُنقذ إلى مادة واحدة فقط. لذا، تُختم المعلومات إلى عدد من **الصفحات** *pages* المتساوية الحجم من البناات، تظهر متتابعة في الصينيات، وكل قراءة أو كتابة في القرص تكون لصفحة كاملة أو لعدة صفحات. ففي قرص غوذجي، يمكن أن يكون طول الصفحة من 2<sup>11</sup> إلى 2<sup>14</sup> بايتًا. بعد أن يأخذ رأس القراءة والكتابة وضعه الصحيح، ويكون القرص قد دار إلى بداية الصفحة المطلوبة، تصبح القراءة على قرص مخمض أو الكتابة عليه إلكترونية تمامًا (باستثناء دوران القرص)، ويمكن للقرص قراءة مقدار كبير من المعطيات أو كتابتها بسرعة. في معظم الأحيان، يأخذ وقت النفاذ إلى صفحة من المعلومات وقراءتها من القرص زمنًا أكبر من الزمن الذي يأخذه الحاسوب لفحص كل المعلومات المقروءة. لهذا السبب، سننظر في هذا الفصل إلى كلٍّ من المكونتين الرئيسيتين لزمن التنفيذ على حدة:

- عدد مرات النفاذ إلى القرص، و
- زمن وحدة المعالجة المركزية CPU (زمن الحساب).

نقيس عدد مرات النفاذ إلى القرص بدلالة عدد صفحات المعلومات التي تلزم قراءتها من القرص أو كتابتها عليه. نلاحظ أن زمن النفاذ إلى القرص ليس ثابتاً -- بل يتعلق بالمسافة بين المسلك الحالي والمسلك المطلوب، كما يتعلق أيضًا بملوضع الدوراني البدني للقرص. ومع ذلك، فإننا سنستخدم عدد الصفحات المقروءة أو المكتوبة باعتباره تقريباً من المرتبة الأولى للزمن الإجمالي المصروف للنفاذ إلى القرص.

في تطبيق نموذجي للأشجار للمعشمة، يكون حجم المعطيات المعالجة ضئيلاً جداً، بحيث لا يمكن وضع كل المعطيات فوراً في الذاكرة الرئيسية. تنسخ خوارزميات الأشجار للمعشمة صفحات مختارة من القرص إلى الذاكرة الرئيسية عند الحاجة إليها، وتعيد كتابة الصفحات التي تغترب على القرص. تحتفظ خوارزميات B-tree بعدد ثابت فقط من الصفحات في الذاكرة الرئيسية في أي وقت؛ لذا لا يتحد حجم الذاكرة الرئيسية من حجم الأشجار للمعشمة التي يمكن معالجتها.

نمذج عمليات القرص في شبه الرماز الخاص بنا كما يلي. ليكن  $x$  مؤشر إلى غرض. فإذا كان الغرض موجوداً حالياً في الذاكرة الرئيسية للحاسوب، أمكننا العودة إلى واصفاته كالمعتاد: على سبيل المثال  $x.key$ . أما إذا كان الغرض الذي يشير إليه  $x$  موجوداً على القرص، فيجب أن ننفذ عملية  $DISK-READ(x)$  لقراءة الغرض  $x$  إلى الذاكرة الرئيسية قبل أن تتمكن من العودة إلى واصفاته. (نفترض أنه إذا كان  $x$  موجوداً سلفاً في الذاكرة الرئيسية، فإن  $DISK-READ(x)$  لا يتطلب نفاذاً إلى القرص؛ وهذا يكافئ "لا عملية no-op"). وبالمثل، تُستخدم عملية  $DISK-WRITE(x)$  لتخزين أي تغييرات كانت قد جرت على واصفات الغرض  $x$ . أي إن الشكل النموذجي للعمل مع غرض ما يكون كالتالي:

$x =$  مؤشر إلى غرض ما

$DISK-READ(x)$

عمليات نفاذ و/أو تعديل على واصفات ■

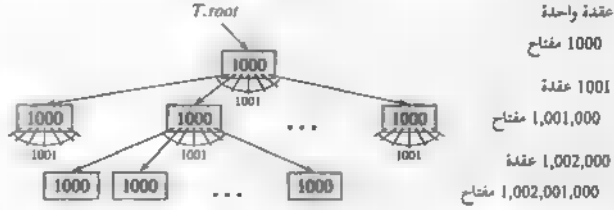
$DISK-WRITE(x)$  // يمكن إغفاله إذا لم يجر أي تعديل على واصفات  $x$

عمليات أخرى تُنفذ إلى واصفات  $x$ ، ولكن دون أن تعُد عليها

يمكن للنظام أن يحتفظ بعدد محدود فقط من الصفحات في الذاكرة الرئيسية في وقت ما. سنفترض أن النظام يُفَرِّج الصفحات التي لم تعد مستخدمة من الذاكرة الرئيسية؛ وستحتاج خوارزميات B-tree الخاصة بنا هذا الأمر.

ولما كان زمن تنفيذ خوارزمية B-tree، في معظم الأنظمة، يعتمد اعتماداً رئيسياً على عدد عمليات القراءة  $DISK-READ$  والكتابة  $DISK-WRITE$  التي تجريها على القرص، فإننا نرغب عادةً في أن تقرأ (أو تكتب) كل من هذه العمليات أكبر قدر ممكن من المعلومات. وبذلك، تكون عقدة شجرة B-tree عادةً كبيرة بقدر صفحة قرص كاملة، وهذا الحجم يحد من عدد أبناء عقدة B-tree.





**الشكل 3.18** شجرة B-tree ارتفاعها 2 تحتوي على أكثر من مليار مفتاح. يظهر في داخل كل عقدة  $x$  عدد المفاتيح  $x.n$  في هذه العقدة. تحتوي كل عقدة داخلية وورقة على 1000 مفتاح. ثمة 1001 عقدة على العمق 1، وأكثر من مليون ورقة على العمق 2.

في حالة شجرة B-tree ضخمة مخزنة على القرص، يكون عامل التفرع بين 50 و2000 غالبًا، وذلك اعتمادًا على نسبة حجم المفتاح إلى حجم الصفحة. يقلص عامل التفرع الكبير كلاً من ارتفاع الشجرة وعدد مرات النفاذ إلى القرص المطلوبة للعثور على أي مفتاح تقليصًا كبيرًا. يُظهر الشكل 3.18 شجرة B-tree بعامل تفرع 1001 وارتفاع 2، يمكنها أن تخزن أكثر من مليار مفتاح؛ مع ذلك، لما كان بإمكاننا الاحتفاظ دومًا بالعقدة الجذر في الذاكرة، فيمكننا العثور على أي مفتاح في هذه الشجرة بإجراء نفاذين إلى القرص على الأكثر.

## 1.18 تعريف الأشجار المعممة

سنفترض، للتبسيط، أن أي معلومات تابعة مرفقة مع مفتاح ما ستكون مخزنة في نفس عقدة المفتاح، كما فعلنا في أشجار البحث الثنائية والأشجار الحمراء-السوداء. عمليًا، يمكننا مع كل مفتاح، تخزين مؤشر يشير إلى صفحة قرص أخرى تتضمن المعلومات التابعة لذلك المفتاح. نفترض شبه الرمز في هذا الفصل ضمنيًا أنه يجري ترحيل للمعلومات التابعة المرفقة مع مفتاح ما، أو للوشر إلى هذه المعلومات، مع المفتاح فيما إذا تحرك المفتاح من عقدة إلى عقدة. ثمة متغير شائع من الأشجار B-trees، يُعرف بـ  $B^+ \text{-trees}$ ، يُخزن كل للمعلومات التابعة في الأوراق ويخزن فقط للمفاتيح ومؤشرات الأبناء في العقد الداخلية، وبذلك يصبح عامل التفرع في العقد الداخلية أعظميًا.

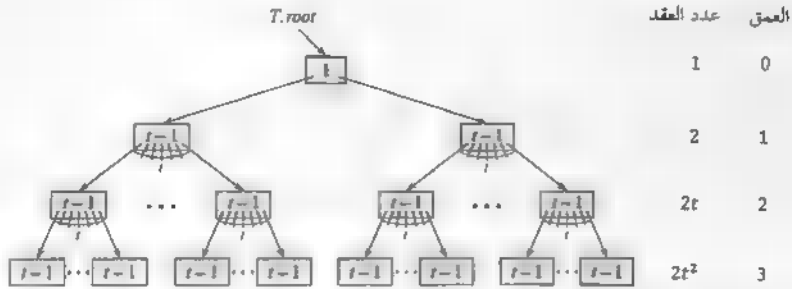
إن شجرة معممة  $T$  هي شجرة ذات جذر (جذرها هو  $T.root$ ) لها الخصائص التالية:

1. لكل عقدة  $x$  فيها الوصفات التالية:

أ.  $x.n$  عدد للمفاتيح المخزنة حاليًا في العقدة  $x$ ،

- ب. المفاتيح  $x.n$  نفسها،  $x.key_1, x.key_2, \dots, x.key_{x.n}$  مخزنة بترتيب غير تنازلي، أي إن:
- $$x.key_1 \leq x.key_2 \leq \dots \leq x.key_{x.n}$$
- ت.  $x.leaf$  هي قيمة منطقية: صح TRUE إذا كانت  $x$  ورقة، وخطأ FALSE إذا كانت  $x$  عقدة داخلية.
2. تحتوي كل عقدة داخلية  $x$  أيضاً  $x.n + 1$  مؤشرًا  $x.c_1, x.c_2, \dots, x.c_{x.n+1}$  إلى أبنائها. وليس للعقد الأوراق أبناء، لذلك فإن واصفات  $c_i$  الخاصة بها غير معرّنة.
3. تُفصل المفاتيح  $x.key_i$  بحالات للمفاتيح للمخزنة في كل شجرة جزئية: إذا كان  $k_i$  مفتاحًا محزّنًا في الشجرة الفرعية ذات الجذر  $x.c_i$  فإن:
- $$k_1 \leq x.key_1 \leq k_2 \leq x.key_2 \leq \dots \leq x.key_{x.n} \leq k_{x.n+1}.$$
4. لكل الأوراق العمق نفسه، وهو ارتفاع الشجرة  $k$ .
5. توجد حدود دنيا وحدود عليا لعدد المفاتيح التي يمكن أن تتضمنها عقدة ما. نعرّف عن هذه الحدود بدلالة عدد صحيح ثابت  $t \geq 2$  يُسمّى **الدرجة الدنيا** *minimum degree* للشجرة B-tree:
- أ. يجب أن يكون لكل عقدة (ما عدا عقدة الجذر)  $t-1$  مفتاحًا على الأقل. لذلك، تتضمن كل عقدة داخلية غير الجذر  $t$  ابنًا على الأقل. إذا لم تكن الشجرة فارغة، يجب أن يكون للجذر مفتاح واحد على الأقل.
- ب. يمكن أن تتضمن كل عقدة  $2t-1$  مفتاحًا على الأكثر. لذلك، يمكن أن يكون لأي عقدة داخلية  $2t$  ابنًا على الأكثر. نقول عن عقدة أنها **ممتلئة** *full* إذا كانت تحوي  $2t-1$  مفتاحًا تمامًا.<sup>2</sup>
- نُحدّث أبسط شجرة B-tree عندما يكون  $t = 2$ . عندئذٍ تحتوي كل عقدة داخلية على اثنين اثنين، أو 3 أبناء، أو 4 أبناء، ويكون لدينا شجرة 2-3-4. ولكن، عمليًا تعطي قيم  $\parallel$  الكبيرة عددًا أشجارًا ممتلئة بارتفاعات أقل.
- ارتفاع شجرة B-tree**
- يتناسب عدد مرات النفاذ إلى القرص الذي تتطلبه معظم العمليات على الأشجار B-trees مع ارتفاع الشجرة. نحلّل الآن ارتفاع شجرة B-tree في أسوأ الأحوال.

<sup>2</sup> ثمة متغير شائع آخر من الأشجار B-tree، يُعرف بـ  $B^+-tree$ ، يتطلب أن تكون كل عقدة داخلية ممتلئة بنسبة  $2/3$  على الأقل، عوضًا عن أن تكون نصف ممتلئة كما تتطلب B-tree.



**الشكل 4.18** شجرة معممة ارتفاعها 3، تحتوي على الحد الأدنى من المفاتيح. يظهر ضمن كل عقدة  $x$  عدد مفاتيحها  $x.n$ .

### مبرنة 1.18

إذا كان  $n \geq 1$ ، فيكون لأي شجرة B-tree  $T$  ذات  $n$  مفاتيح، وارتفاع  $h$ ، ودرجة دنيا  $t \geq 2$ :

$$h \leq \log_t \frac{n+1}{2}.$$

**البرهان** يتضمن جذر شجرة B-tree  $T$  مفاتيحاً واحداً على الأقل، وتتضمن كل العقد الأخرى  $t-1$  مفاتيحاً على الأقل. إذن، يوجد للشجرة  $T$  ذات الارتفاع  $h$ ، عقدتان على الأقل على عمق 1، و  $2t$  عقدة على الأقل على عمق 2، و  $2t^2$  عقدة على الأقل على عمق 3، وهكذا... إلى أن نصل إلى العمق  $h$ ، حيث يوجد  $2t^{h-1}$  عقدة على الأقل. يوضح الشكل 4.18 هذه الشجرة في حالة  $h = 3$ . إذن يحقق العدد  $n$  من المفاتيح للمتراحة التالية:

$$\begin{aligned} n &\geq 1 + (t-1) \sum_{i=1}^h 2t^{i-1} \\ &= 1 + 2(t-1) \left( \frac{t^h - 1}{t - 1} \right) \\ &= 2t^h - 1. \end{aligned}$$

وبعمليات جبرية بسيطة نحصل على المتراحة  $(n-1)/2 \leq t^h$ . وبأخذ اللوغاريتم ذي الأساس  $t$  للطرفين نكون قد برهننا النظرية. ■

نرى هنا قوة الأشجار B-trees مقارنةً بالأشجار الحمراء-السوداء. ومع أن ارتفاع الشجرة يزداد بدرجة  $O(\lg n)$  في كلتا الحالتين (تذكر أن  $t$  ثابت)، إلا أن أساس اللوغاريتم في الأشجار للمعممة يمكن أن يكون أكبر بعدة مرات. لذلك، توفر الأشجار للمعممة  $\lg t$  تقريباً على عدد العقد المتفحصة في معظم عمليات

الأشجار، مقارنة بالأشجار الحمراء-السوداء. ولما كان من اللازم عادةً النفاذ إلى القرص لتفحص عقدة ما في شجرة، فإنَّ الأشجار للمُعَمَّمة تنفّادى عددًا كبيرًا من مرات النفاذ إلى القرص.

تعاريف

1-1.18

لم لا نسمح بدرجة أصغر من  $e = 1$ ؟

2-1.18

ما هي قيم  $e$  التي تكون فيها الشجرة في الشكل 1.18 شجرةً معُمَّمةً صحيحة؟

3-1.18

أظهر جميع الأشجار للمُعَمَّمة الصحيحة من الدرجة الصغرى 2 التي تمثل  $\{1, 2, 3, 4, 5\}$ .

4-1.18

ما هو العدد الأعظم للمفاتيح التي يمكن تخزينها في شجرة معُمَّمة ذات ارتفاع  $h$  بدلالة الدرجة الصغرى  $e$ ؟

5-1.18

صِف بنية المعطيات الناتجة إذا امتصت كل عقدة سوداء في شجرة حمراء-سوداء أبناءها الأحمر وضُمَّت أبناءها إلى أبناءها.

## 2.18 العمليات الأساسية على الأشجار المعُمَّمة

نقدم في هذا المقطع تفاصيل عمليات البحث B-TREE-SEARCH والإنشاء B-TREE-CREATE والإدخال B-TREE-INSERT على الأشجار المعُمَّمة. نعتمد في هذه الإجراءات اصطلاحين:

- جذر الشجرة للمُعَمَّمة هو دومًا في الذاكرة الرئيسية، لذلك لا نحتاج أبدًا إلى تنفيذ عملية قراءة من القرص DISK-READ للبحث؛ ولكن علينا تنفيذ عملية كتابة على القرص للجذر DISK-WRITE عندما نغير عقده.

- يجب تنفيذ عملية قراءة مسبقة DISK-READ لأي عقدة تمرّ على أيّ أعماموسطات.

إن جميع الإجراءات التي تقدمها هي خوارزميات يمرور واحد "one-pass" التي تتقدم نزولاً من جذر الشجرة دون الحاجة إلى الرجوع خلفًا.

### البحث في شجرة معُمَّمة

يشبه البحث في شجرة معُمَّمة كثيرًا البحث في شجرة ثنائية، إلا أننا بدلاً من اتخاذ قرار تفريع ثنائي ذي

طريقتين، نتخذ قراراً تفريعاً متعدد الطرق بحسب عدد أولاد العقدة. وبعبارة أدق، عند كل عقدة داخلية  $x$ ، نتخذ قراراً تفريعاً بـ  $(x, n+1)$  طريقاً.

إن الإجراء B-TREE-SEARCH هو تصميم مباشر للإجراء TREE-SEARCH المعروف على الأشجار الثنائية. دخل الإجراء B-TREE-SEARCH هو المؤشر  $x$  إلى عقدة جذر في شجرة جزئية، والمفتاح  $k$  الذي يجب البحث عنه في هذه الشجرة الفرعية. لذلك، فإن الاستدعاء على المستوى الأعلى هو من الشكل  $(y, i)$  B-TREE-SEARCH( $T, root, k$ ). فإذا كان  $k$  موجوداً في الشجرة، فإن هذا الإجراء يعيد الزوج المرتب  $(y, i)$  الذي يتألف من العقدة  $y$  ودليل  $i$  بحيث يكون  $y.key_i = k$ . وإلا، فإنه يعيد القيمة NIL.

B-TREE-SEARCH( $x, k$ )

```

1   $i = 1$ 
2  while  $i \leq x.n$  and  $k > x.key_i$ 
3       $i = i + 1$ 
4  if  $i \leq x.n$  and  $k == x.key_i$ 
5      return  $(x, i)$ 
6  elseif  $x.leaf$ 
7      return NIL
8  else DISK-READ( $x, c_i$ )
9      return B-TREE-SEARCH( $x, c_i, k$ )

```

وباستخدام إجراء بحث خطي، تعثر الأسطر 3-1 على الدليل الأصغر  $i$  الذي يحقق  $k \leq x.key_i$ ، أو أنها تجعل قيمة  $i$  مساوية  $x.n + 1$ . تنفحص الأسطر 4-5 اكتشاف المفتاح، وتعيده إذا اكتشفته. وإلا فإن الأسطر 6-9 تنهي البحث بالإخفاق (إذا كانت  $x$  ورقة) أو تطلب تنفيذاً عودياً للبحث ضمن الشجرة الفرعية المناسبة لـ  $x$ ، بعد إجراء القراءة الضرورية من القرص DISK-READ على ذلك الابن.

يوضح الشكل 1.18 عملية البحث B-TREE-SEARCH؛ يتفحص الإجراء العقدة المظلمة تظليلاً خفيفاً أثناء البحث عن المفتاح  $R$ .

كما في حالة الإجراء TREE-SEARCH لأشجار البحث الثنائية، تُشكّل العقد التي جرت ملاقاتها خلال الإجراء القوّدي طريقاً بسيطاً نازلاً من جذر الشجرة. ولذا، يُنفَّذ الإجراء B-TREE-SEARCH إلى  $O(h) = O(\log_2 n)$  صفحة من القرص، حيث  $h$  هو ارتفاع الشجرة الممتعة و  $n$  هو عدد المفاتيح فيها. وما كان  $x.n < 2t$ ، فإنّ حلقة While في السطرين 2-3 تأخذ زمناً ضمن كل عقدة هو  $O(t)$ ، ويكون الزمن الإجمالي لوحدة المعالجة المركزية هو  $O(t \log_2 n)$ .

إنشاء شجرة ممتعة فارغة

لبناء شجرة ممتعة  $T$ ، نستخدم أولاً B-TREE-CREATE لإنشاء عقدة جذر فارغة، ثم نستدعي

**B-TREE-INSERT** لإضافة مفاتيح جديدة. يستخدم كلٌّ من هذين الإجراءين إجراءً مساعدًا **ALLOCATE-NODE** يُخصّص صفحةً واحدةً في القرص لاستخدامها كعقدة جديدة في زمن  $O(1)$ . يمكن أن نفترض أن العقدة المنشأة باستخدام **ALLOCATE-NODE** لا تتطلب قراءة **DISK-READ**، لأن القرص لا يتضمن بعدُ معلوماتٍ مفيدةً مخزنةً على القرص عن هذه العقدة.

#### B-TREE-CREATE( $T$ )

```

1   $x = \text{ALLOCATE-NODE}()$ 
2   $x.\text{leaf} = \text{TRUE}$ 
3   $x.n = 0$ 
4  DISK-WRITE( $x$ )
5   $T.\text{root} = x$ 

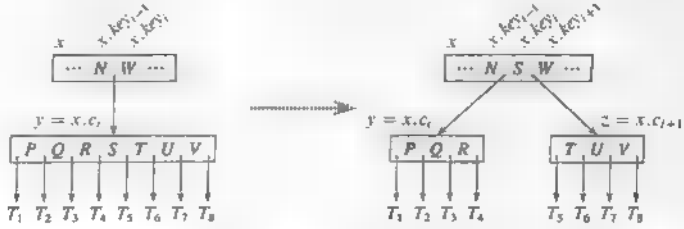
```

إن الإجراء **B-TREE-CREATE** يتطلب  $O(1)$  عملية على القرص وزمنًا  $O(1)$  من وحدة المعالجة المركزية CPU.

#### إدراج مفتاح في شجرة معيَّمة

إن إدراج مفتاح في شجرة معيَّمة هو أعقد بكثير من إدراج مفتاح في شجرة بحث ثنائية. كما في حالة أشجار البحث الثنائية، نبحث عن موضع الورقة التي يجب إدراج المفتاح الجديد فيها. لكننا، في الشجرة المعيَّمة لا نستطيع ببساطة إنشاء عقدة ورقة جديدة وإدراجها، لأن الشجرة الناتجة عندها قد تُخفق في أن تكون شجرة معيَّمة صالحة. عوضًا عن ذلك، فإننا ندرج المفتاح الجديد في عقدة ورقة موجودة. ولما كنا لا نستطيع إدراج مفتاح في عقدة ورقة ملأنة، فإننا نُدخل عملية **تفريق** *splitting* لعقدٍ ملأنة  $y$  (تتضمن  $2t - 1$  مفتاحًا) حول **مفتاحها الوسط**  $median\ key\ y.\text{key}_t$  إلى عقدتين تتضمن كل منهما  $t - 1$  مفتاحًا. يتحرك المفتاح الوسط إلى الأعلى ضمن العقدة الأب لـ  $y$  ليحدد نقطة التقسيم بين الشجرتين الجديدتين. ولكن، إذا كانت العقدة الأب الخاصة بـ  $y$  ملأنة أيضًا، فيجب أن نفرِّقها قبل أن نتسكن من إدراج المفتاح الجديد، ومن ثم فإننا نستطيع إنهاء تفريق العقد الملأنة على كامل الطريق إلى أعلى الشجرة.

وكما في شجرة البحث الثنائية، يمكننا إدراج مفتاح في شجرة معيَّمة بمرور واحد نزولاً ضمن الشجرة من الجذر إلى ورقة. ولعمل ذلك لا ننتظر لمعركة حاجتنا الفعلية لتفريق عقدة ملأنة لكي ننفِّذ الإدراج. بل نقوم - أثناء انتقالنا نزولاً في الشجرة - بالبحث عن الموضع الذي ينتمي إليه المفتاح الجديد - بتفريق كل عقدة ملأنة نصل إليها في الطريق (ومن ضمنها الورقة نفسها). وبذلك عندما نريد تفريق عقدة ملأنة  $y$ ، نكون على يقين بأن العقدة الأب الخاصة بها غير ملأنة.



**الشكل 5.18** تفريق عقدة في حالة  $t = 4$ . جرى تفريق العقدة  $y = x.c_t$  إلى عقدتين  $y$  و  $z$ ، وانتقال المفاتيح الوسط  $S$  في  $y$  إلى الأعلى ضمن العقدة الأب.

### تفريق عقدة في شجرة معشمة

دخّل الإجراء **B-TREE-SPLIT-CHILD** هو عقدة داخلية غير ممثلة  $x$  (نفترض أنها في الذاكرة الرئيسية)، ودليل  $i$ ، وعقدة  $y$  (نفترض أنها في الذاكرة الرئيسية أيضًا) بحيث تكون  $x.c_i$  ابنًا ممثلًا لـ  $x$ . بعد ذلك، يقوم الإجراء بتفريق هذا الابن إلى اثنين، وضبط  $x$  بحيث يصبح له ابنٌ إضافي. ونفريق جذر ملآن، نجعل هذا الجذر أولًا ابنًا لعقدة جذر جديدة فارغة بحيث نستطيع استخدام **B-TREE-SPLIT-CHILD**. ومن ثم فإن الشجرة يزيد ارتفاعها بمقدار واحد؛ فالتفريق هو الطريقة الوحيدة التي تنمو بها الشجرة.

يوضّح الشكل 5.18 هذا الإجراء. نفرّق العقدة المملّنة  $y = x.c_i$  حول مفاتيحها الوسط  $S$  الذي ينتقل إلى الأعلى إلى العقدة الأب  $x$ . جرى وضع المفاتيح الموجودة في  $y$  ذات القيمة التي هي أكبر من الوسط ضمن عقدة جديدة  $z$ ، التي أصبحت ابنًا جديدًا لـ  $x$ .

#### **B-TREE-SPLIT-CHILD**( $x, i$ )

```

1   $z = \text{ALLOCATE-NODE}()$ 
2   $y = x.c_i$ 
3   $z.\text{leaf} = y.\text{leaf}$ 
4   $z.n = t - 1$ 
5  for  $j = 1$  to  $t - 1$ 
6       $z.\text{key}_j = y.\text{key}_{j+t}$ 
7  if not  $y.\text{leaf}$ 
8      for  $j = 1$  to  $t$ 
9           $z.c_j = y.c_{j+t}$ 
10  $y.n = t - 1$ 
11 for  $j = x.n + 1$  downto  $i + 1$ 
12      $x.c_{j+1} = x.c_j$ 
13  $x.c_i + 1 = z$ 
14 for  $j = x.n$  downto  $i$ 
```

```

15   x.keyj+1 = x.keyj
16   x.keyi = y.keyi
17   x.n = x.n + 1
18   DISK-WRITE(y)
19   DISK-WRITE(z)
20   DISK-WRITE(x)

```

يعمل الإجراء B-TREE-SPLIT-CHILD مباشرة بطريقة الفص واللصق. العقدة  $\equiv$  هنا هي العقدة التي يجري تفريقها، والعقدة  $y$  هي الابن  $i$  للعقدة  $x$  (وُضِعت في السطر 2). تتضمن العقدة  $y$  أصلاً  $2t$  ابنًا ( $2t-1$  مفتاحًا) ولكن جرى تقليصها بهذه العملية إلى  $t$  ابنًا ( $t-1$  مفتاحًا). تأخذ العقدة  $z$  الأبناء  $t$  الكبار ( $t-1$  مفتاحًا) في  $y$ ، وتصبح ابنًا جديدًا لـ  $x$ ، وتوضع مباشرة بعد  $y$  في جدول أبناء  $x$ . ينتقل المفتاح الوسيط في  $y$  إلى الأعلى ليصبح للمفتاح الذي يفصل بين  $y$  و  $z$  في  $x$ .

تُنشئ الأسطر 9-1 عقدة  $\equiv$  وتعطيها المفاتيح  $t-1$  الأكبر والأولاد الموافقين لها في  $y$ . يصبح السطر 10 عدد المفاتيح في  $y$ . أخيرًا تُدرج الأسطر 11-17 العقدة  $z$  باعتبارها ابنًا جديدًا لـ  $x$ ، وتنقل المفتاح الوسيط من  $y$  إلى  $x$  بهدف فصل  $y$  عن  $z$ ، ثم تُصَحَّح عدد المفاتيح في  $x$ . الأسطر 18-20 تعيد كتابة صفحات القرص المعثلة. إن زمن وحدة المعالجة المركزية الذي يستخدمه هذا الإجراء B-TREE-SPLIT-CHILD هو  $\Theta(t)$ ، بسبب الحلقات في الأسطر 5-6 و 8-9. (الحلقات الأخرى يجري تنفيذها بـ  $O(t)$  تكرارًا.) يقوم الإجراء بـ  $O(1)$  عملية على القرص.

**إدخال مفتاح في شجرة معثمة بمرور واحد نزولاً عبر الشجرة**

تُدْرَج مفتاحًا  $k$  في شجرة معثمة  $T$  ذات ارتفاع  $h$  بمرور واحد نزولاً عبر الشجرة، وهذا يتطلب  $O(h)$  نفاذًا إلى القرص. أما وحدة المعالجة المركزية، فتتطلب زمنًا  $O(t \log_e n) = O(t \log_e n)$ . ويُستخدم الإجراء B-TREE-INSERT لإجراء B\_TREE\_SPLIT\_CHILD لضمان عدم نزول القودية إلى عقدة معثلة.

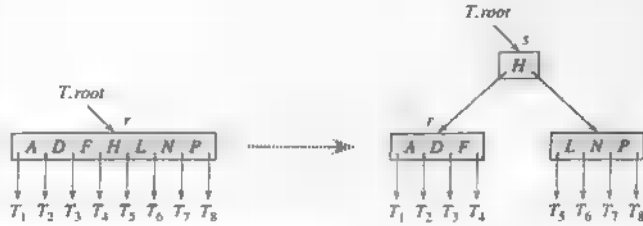
**B-TREE-INSERT( $T, k$ )**

```

1   r = T.root
2   if r.n == 2t - 1
3       s = ALLOCATE-NODE()
4       T.root = s
5       s.leaf = FALSE
6       s.n = 0
7       s.c1 = r
8       B-TREE-SPLIT-CHILD(s, 1)
9       B-TREE-INSERT-NONFULL(s, k)
10  else B-TREE-INSERT-NONFULL(r, k)

```





**الشكل 6.18** تفريق الجذر في حالة  $t = 4$ . يتفرق الجذر  $r$  إلى عقدتين، وأنشئ جذر جديد  $s$ . يحتوي الجذر الجديد المفتاح الوسط في  $r$  ويحتوي نصفي  $r$  بوصفهما ولذين. تنمو الشجرة الممتعة بزيادة ارتفاعها بمقدار واحد عند تفريق الجذر.

تُعالج الأسطر 3-9 الحالة التي يكون فيها الجذر  $r$  ممتلئاً: يتفرق الجذر، وتُنشئ عقدة جديدة  $s$  (لها ابنان) هذا الجذر. إن تفريق الجذر هي الطريقة الوحيدة لزيادة ارتفاع الشجرة الممتعة. يُظهر الشكل 6.18 هذه الحالة. وخلافاً لشجرة البحث الثنائية، يزداد ارتفاع الشجرة الممتعة في قمتها بدلاً من أسفلها. ينتهي الإجراء باستدعاء الإجراء **B-TREE-INSERT-NONFULL** للقيام بإدخال مفتاح  $k$  في الشجرة ذات الجذر غير الممتلئ. ويقوم الإجراء **B-TREE-INSERT-NONFULL** بتنفيذ عودي نزولاً في الشجرة حسب الضرورة، وفي كل الأوقات، يضمن ألا تكون العقدة التي يعود إليها ممتلئة باستدعاء **B\_TREE\_SPLIT\_CHILD** عند الضرورة.

يُدرج الإجراء العودي المساعد **B-TREE-INSERT-NONFULL** مفتاحاً  $k$  ضمن عقدة  $x$  يفترض أن تكون غير ممتلئة عند طلب الإجراء. تُضمن عملية **B-TREE-INSERT** والعملية العودية **B-TREE-INSERT-NONFULL** صحيحة هذا الافتراض.

**B-TREE-INSERT-NONFULL( $x, k$ )**

```

1   $i = x.n$ 
2  if  $x.leaf$ 
3      while  $i \geq 1$  and  $k < x.key_i$ 
4           $x.key_{i+1} = x.key_i$ 
5           $i = i - 1$ 
6       $x.key_{i+1} = k$ 
7       $x.n = x.n + 1$ 
8      DISK-WRITE( $x$ )
9  else while  $i \geq 1$  and  $k < x.key_i$ 
10      $i = i - 1$ 
11      $i = i + 1$ 
12     DISK-READ( $x.c_i$ )
    
```

```

13  if  $x.c_i.n == 2t - 1$ 
14      B-TREE-SPLIT-CHILD( $x, i$ )
15      if  $k > x.key_i$ 
16           $i = i + 1$ 
17  B-TREE-INSERT-NONFULL( $x, c_i, k$ )

```

يعمل إجراء B-TREE-INSERT-NONFULL كما يلي. تعالج الأسطر 3-8 الحالة التي تكون فيها العقدة  $x$  ورقة بإدراج المفتاح  $k$  ضمن  $x$ . فإذا لم تكن  $x$  ورقة، فيجب إدراج  $k$  في الورقة المناسبة من الشجرة الفرعية التي يكون جذرها العقدة الداخلية  $x$ . في هذه الحالة، تُحدّد الأسطر 9-11 ابن  $x$  الذي ينزل إليه التنفيذ العودي. يتحرّى السطر 13 نزول التنفيذ العودي إلى عقدة متعلقة، وفي هذه الحالة يستخدم السطر 14 الإجراء B-TREE-SPLIT-CHILD لتفريق الابن إلى ابنين غير متعلقين، وتُحدّد السطران 15-16 أيّ الابنين هو الآن العقدة الصحيحة التي يجب النزول إليها. (لاحظ أنه لا حاجة للإجراء DISK-READ( $x.c_i$ ) بعد أن يزيد السطر 16 قيمة  $i$  بواحد، لأن الإجراء العودي سينزل في هذه الحالة إلى ابن أنشأه للتو B-TREE-SPLIT-CHILD). إن الأثر الصافي الناتج عن الأسطر 13-16 هو إذن ضمان عدم وصول التنفيذ العودي أبداً إلى عقدة متعلقة. يقوم السطر 17 بعد ذلك بتنفيذ عودي لإدراج  $k$  في الشجرة الفرعية المناسبة. يوضّح الشكل 7.18 حالات الإدراج المختلفة في شجرة معشّمة.

يقوم الإجراء B-TREE-INSERT-NONFULL بـ  $O(h)$  نفاذاً إلى القرص في حالة شجرة معشّمة بارتفاع  $h$ ، لأنه جرى تنفيذ عمليتي DISK-READ و DISK-WRITE  $O(1)$  فقط بين استدعاءات B-TREE-INSERT-NONFULL. أما زمن وحدة المعالجة المركزية الإجمالي للمستخدم، فهو  $O(t \log_e n) = O(t h)$ . ولما كان B-TREE-INSERT-NONFULL عَوْدِيّ الذيل tail-recursive، فيمكن تحيزه تحيزاً بديلاً باستخدام حلقة while، وهذا يبرهن أن عدد الصفحات التي يلزم بقاؤها في الذاكرة الرئيسية في أي وقت هي  $O(1)$ .

تمارين

1-2.18

أظهر نتائج إدراج المفاتيح

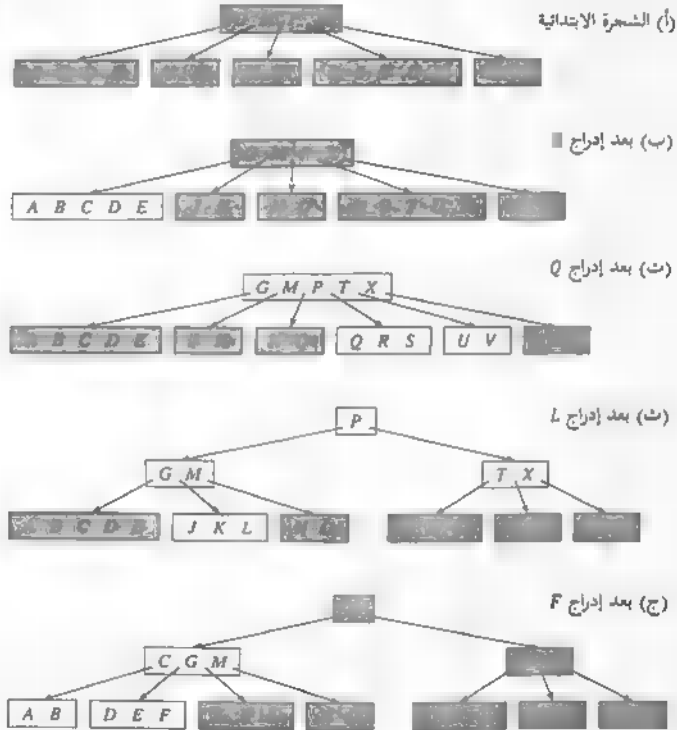
$F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E$

بالترتيب من اليسار إلى اليمين في شجرة معشّمة فارغة درجتها الصغرى 2. ارسم شكل الشجرة قبل وجوب تفريق عقدة ما فقط، وارسم كذلك شكلها النهائي.

2-2.18

وضّح تحت أية ظروف، إن وحدت، تكون عمليات DISK-READ و DISK-WRITE حشواً خلال تنفيذ استدعاء للإجراء B-TREE-INSERT. (تكون عملية DISK-READ حشواً إذا جرت لصفحة موجودة مسلفاً

في الذاكرة. وتكون عملية DISK-WRITE حشواً إذا جرت كتابة صفحة من المعلومات على القرص وكانت هذه الصفحة مطابقة لما هو مخزن فيها سابقاً.)



**الشكل 7.18** إدراج مفاتيح في شجرة معشمة. الدرجة الصغرى 3 في هذه الشجرة المعشمة هي 3، لذلك يمكن أن تحتوي عقدة ما 5 مفاتيح على الأكثر. العقد للظلال نظلياً خفيفاً هي العقد التي عدّها إجراء الإدراج. (أ) الشجرة الابتدائية في هذا المثال. (ب) نتيجة إدراج B في الشجرة الابتدائية؛ هذا إدراج بسيط في ورقة. (ت) نتيجة إدراج Q في الشجرة السابقة. تفرّقت العقدة RSTUV إلى عقدين تحتويان RS و UV، ويُكَبَّل المفتاح T إلى الجذر في الأعلى، وأُدرج Q في النصف الأيسر (العقدة RS). (ث) نتيجة إدراج L في الشجرة السابقة. تفرّق الجذر مباشرةً لأنه ممتلئ، وزاد ارتفاع الشجرة بمقدار واحد. ثم أُدرج L في الورقة التي تحتوي JK. (ج) نتيجة إدراج F في الشجرة السابقة. تفرّقت العقدة ABCDE قبل إدراج F في النصف الأيمن (العقدة DE).

### 3-2.18

اشرح كيف يمكن إيجاد المفتاح الأصغر المخزن في شجرة معشمة، وكيف يمكن إيجاد المفتاح السابق لمفتاح معين مخزن في شجرة معشمة.

### \* 4-2.18

افترض أننا أدرجنا المفاتيح  $\{1, 2, \dots, n\}$  في شجرة معشمة فارغة درجتها الصغرى 2. ما عدد العقد في الشجرة المعممة النهائية؟

### 5-2.18

لما كانت العقد الأوراق لا تتطلب مؤشرات إلى الأبناء، فيمكن لها من حيث المبدأ استخدام قيمة مختلفة ل  $e$  (أكبر منها) عن تلك الخاصة بالعقد الداخلية في حالة الحجم نفسه من صفحة القرص. برهن كيف يمكن تعديل إجراءات الإنشاء والإدراج في شجرة معشمة لمعالجة هذا الاختلاف.

### 6-2.18

افترض أننا نريد تنحيز B-TREE-SEARCH باستخدام بحث ثنائي بدلاً عن البحث الخطي ضمن كل عقدة. برهن أن هذا التغيير يجعل الزمن المطلوب من وحدة للمعالجة المركزية  $O(\lg n)$ ، بمعزل عن كيفية اختيار  $e$  باعتباره دالة ل  $n$ .

### 7-2.18

افترض أن عتادات القرص تسمح لنا باختيار حجم صفحة القرص بحرية، لكن الزمن اللازم لقراءة صفحة من القرص هو  $a + bt$ ، حيث  $a$  و  $b$  ثابتان محدّدان و  $t$  هي الدرجة الصغرى لشجرة معشمة تستخدم صفحات من الحجم المختار. صف كيفية اختيار  $t$  بحيث يكون زمن البحث في الشجرة للمعشمة أصغر (تقريباً). اقترح قيمة أمثلة ل  $e$  في الحالة التي تكون فيها  $a = 5$  ميلي ثانية و  $b = 10$  ميلي ثانية.

## 3.18 حذف مفتاح من شجرة معشمة

يشبه الحذف من شجرة معشمة الإدراج فيها، ولكنه أعقد بقليل، لأنه يمكن حذف مفتاح من أية عقدة - وليس من الأوراق فقط - ويتطلب الحذف من عقدة داخلية إعادة ترتيب أولادها. وكما في حالة الإدراج، يجب أن نحترس من الحذف الذي ينتج عنه شجرة تحتك بنيتها خواصّ الأشجار للمعشمة. ومثلما كان علينا ضمان عدم تضخم عقدة ما تضخمًا كبيرًا بسبب الإدراج، علينا ضمان ألا تصبح عقدة ما صغيرة جدًا بسبب الحذف (باستثناء أن يسمح للحظر بأن يتضمن أقل من العدد الأصغر للمفاتيح  $e-1$ ). وكما أن خوارزمية الإدراج البسيطة يمكن أن تتراجع إذا كانت إحدى العقد على الطريق إلى للكان الذي سيحري إدراج المفتاح فيه ملانة، فإنّ النهج البسيط للحذف يمكن أن يتراجع إذا كانت عقدة ما (باستثناء الجذر)

على الطريق إلى المكان الذي سيجري حذف المفتاح منه تحتوي على الحد الأدنى من المفاتيح.

يُحذف الإجراء B-TREE-DELETE المفتاح  $k$  من الشجرة الفرعية ذات الجذر  $x$ . نصمّم هذا الإجراء ليضمن أنّه كلما استدعيّ عودياً على عقدة  $x$ ، فإنّ عدد المفاتيح في  $x$  يساوي على الأقل الدرجة الصغرى  $t$ . لاحظ أن هذا الشرط يتطلب مفتاحاً إضافياً على عدد المفاتيح الأدنى المطلوب في الشروط المعتادة للشجرة الممتعة، لذلك قد يلزم أحياناً نقل مفتاح ما إلى عقدة ابن قبل نزول التنفيذ العودي إليها. يسمح لنا هذا الشرط الملقّى بحذف مفتاح من الشجرة بمرور واحد نزولاً، دون الحاجة إلى التراجع (باستثناء تراجع واحد سنشرحه). يجب تفسير التوصيف الآتي للحذف من شجرة ممتعة علماً بأنه إذا أصبح الجذر  $x$  عقدة داخلية بدون مفاتيح (يحصل ذلك في الحالتين 2.ت و 3.ب الآتيتين)، نحذف  $\blacksquare$  وأصبح  $x.c_1$ ، الابن الوحيد لـ  $x$ ، جذراً جديداً للشجرة، وهذا يُقصر ارتفاع الشجرة بمقدار واحد ويحافظ على خاصية أن يتضمن جذر الشجرة مفتاحاً واحداً على الأقل (إلا إذا كانت الشجرة فارغة).

سنستعرض كيفية عمل الحذف بدلاً من عرض شبه الرماز. يوضّح الشكل 8.18 الحالات المختلفة لحذف مفاتيح من شجرة ممتعة.

1. إذا كان المفتاح  $k$  في عقدة  $x$  وكانت  $x$  ورقة، فاحذف المفتاح  $k$  من  $x$ .

2. إذا كان المفتاح  $k$  في عقدة  $x$  وكانت  $x$  عقدة داخلية، فافعل الآتي.

أ. إذا كان لابن  $y$  الذي يسبق  $k$  في العقدة  $x$ ، مفتاحاً على الأقل، فأوجد المفتاح السابق لـ  $k$  وهو  $k'$  في الشجرة الفرعية ذات الجذر  $y$ . احذف  $k'$  عودياً، واستعض عن  $k'$  بـ  $k$  في  $x$ . (يمكننا إيجاد  $k'$  وحذفه في مرور وحيد نزولاً.)

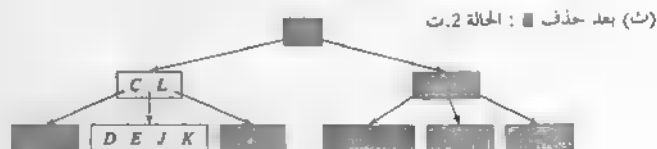
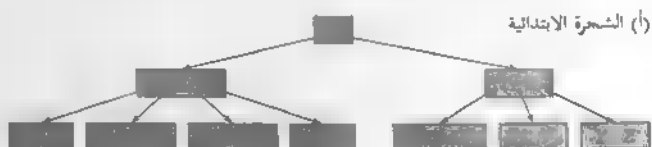
ب. بالتناظر، إذا كان لابن  $y$  عددٌ من المفاتيح أقل من  $t$ ، فتفتح الابن  $z$  الذي يتبع  $k$  في العقدة  $x$ ، إذا كان الابن  $z$  له  $t$  مفتاحاً على الأقل، فأوجد المفتاح التالي لـ  $k$  وهو  $k'$  في الشجرة الفرعية ذات الجذر  $z$ . احذف  $k'$  عودياً، واستعض عن  $k'$  بـ  $k$  في  $x$ . (يمكننا إيجاد  $k'$  وحذفه في مرور وحيد نزولاً.)

ت. فيما عدا ذلك، إذا كان لدى كل من  $y$  و  $z$  فقط  $t-1$  مفتاحاً، فادمج  $k$  وكل مفاتيح  $z$  في  $y$ ، بحيث تفقد  $x$  كلاً من  $k$  والمؤشر إلى  $z$ . تتضمن  $y$  الآن  $2t-1$  مفتاحاً. ثم حرّر  $z$  واحذف  $k$  عودياً من  $y$ .

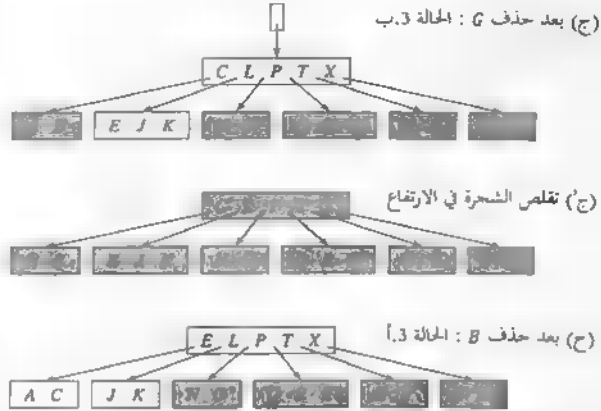
3. إذا لم يكن المفتاح  $k$  موجوداً في عقدة داخلية  $x$ ، حدّد الجذر  $x.c_i$  من الشجرة الفرعية المناسبة الذي يجب أن يحتوي  $k$ ، إذا كان  $k$  موجوداً في الشجرة أصلاً. إذا احتوى  $x.c_i$  فقط  $t-1$  مفتاحاً، فننّذ الخطوة 3.أ أو 3.ب بحسب الضرورة لنضمن أننا ننزل إلى عقدة تحتوي  $t$  مفتاحاً على الأقل. ثم نهي تنفيذ عودي على الابن المناسب لـ  $x$ .

أ. إذا كان  $x, c_i$  ل  $x, c_i$ ،  $t-1$  مفتاحاً، لكن له أخ مباشر يتضمن  $t$  مفتاحاً على الأقل، فأعط  $x, c_i$  مفتاحاً إضافياً بنقل مفتاح من  $x$  هبوطاً إلى  $x, c_i$ ، ونقل مفتاح من الأخ المباشر اليميني أو اليساري ل  $x, c_i$  إلى  $x$ ، ونقل مؤشر الابن المناسب من الأخ إلى  $x, c_i$ .

ب. إذا تضمن كل من  $x, c_i$  وأخويه  $t-1$  مفتاحاً، فادمج  $x, c_i$  مع أحد إخوته، وهذا يتطلب نقل مفتاح من  $x$  إلى الأسفل في العقدة الجديدة المدججة ليصبح للمفتاح الوسط لهذه العقدة.



الشكل 18.8 حذف مفاتيح من شجرة معقمة. الدرجة الصغرى لهذه الشجرة للمعقمة  $t=3$ ، لذلك لا يمكن لأية عقدة غير الجذر أن تحتوي أقل من مفاتيحين. العقد للعلّة هي المظلة نظلياً خفيفاً. (أ) الشجرة المعقمة في الشكل 7.18 (ج). (ب) حذف F. هذه الحالة 1: حذف بسيط من ورقة. (ت) حذف M. هذه الحالة 2.أ: للفتاح السابق ل M وهو L نُقل إلى الأعلى ليأخذ مكان M. (ث) حذف G. هذه الحالة 2.ب: دُفع G إلى الأسفل ليكون العقدة DEGIK، ثم حذف G من هذه الورقة (الحالة 1).



يُتبع الشكل 18.8 (ج) حذف D. هذه الحالة 3.ب: لا يمكن أن ينزل التنفيذ العودي إلى العقدة CL لأن لديها مفتاحين فقط، لذلك دُفع P إلى الأسفل ودمج مع CL و TX ليُشكل CLPTX؛ ثم حذف D من ورقة (الحالة 1). (ج) بعد (ج) حذف الجذر وتقلصت الشجرة في الارتفاع بمقدار واحد. (ح) حذف B. هذه الحالة 3.أ: نُقل C ليمسأ مكان B ونُقل E ليمسأ مكان C.

لما كانت معظم المفاتيح في شجرة معتمدة هي في الأوراق، فيمكن أن نتوقع عملياً أن تُستخدم عمليات الحذف، في غالب الأحيان، لحذف مفاتيح من الأوراق. عندئذٍ يعمل الإجراء B-TREE-DELETE بمرور واحد نزولاً عبر الشجرة دون الحاجة إلى الرجوع. ولكن، عند حذف مفتاح من عقدة داخلية، فإنه يقوم بمرور نزولي عبر الشجرة، ولكن يمكن أن يحتاج إلى العودة إلى العقدة التي لحذف منها المفتاح للاستعاضة عن المفتاح بالمفتاح السابق له أو التالي له (الحالتان 2.أ و 2.ب).

ومع أن هذا الإجراء يبدو معقداً، إلا أنه يتطلب  $O(h)$  عملية على القرص فقط، في حالة شجرة معتمدة ارتفاعها  $h$ ، لأن هنالك استدعاءات لـ DISK-READ و DISK-WRITE من الرتبة  $O(1)$  فقط بين الاستدعاءات العودية للإجراء. أما الزمن المطلوب من وحدة المعالجة المركزية، فهو  $O(t \log_e n) = O(t h)$ .

### تمارين

#### 1-3.18

أظهر نتيجة حذف C و P و V بالترتيب من الشجرة في الشكل 8.18 (ح).

#### 2-3.18

اكتب شبه الرماز الخاص بالإجراء B-TREE-DELETE.

## مسائل

### 1-18 مكملات في الخزن الثانوي

لندرس تنجيز مكس في حاسوب يتضمن مقداراً صغيراً نسبياً من الذاكرة الرئيسية السريعة ومقداراً كبيراً نسبياً من الخزن على القرص الأكثر بطئاً. إن عمليات الدفع PUSH والسحب POP فيه تعمل على قِيم بطول كلمة واحدة. يمكن أن ينمو للمكس الذي نأمل دعمه ليصبح أكبر بكثير مما يمكن أن تتسع له الذاكرة، ومن ثم فإن معظمه يجب أن يكون مخزناً على القرص.

ثمة تنجيز بسيط للمكس، لكنه غير فعال، يحتفظ به كاملاً على القرص. نحتفظ في الذاكرة بمؤشر إلى المكس، يمثل العنوان على القرص للعنصر العلوي في المكس. إذا كانت قيمة للمؤشر  $p$ ، كان العنصر العلوي هو الكلمة ذات الترتيب  $p$  بالملف  $m$  في الصفحة  $[p/m]$  من القرص، حيث  $m$  هو عدد الكلمات في الصفحة.

لتنجيز عملية الدفع PUSH، نزيد مؤشر المكس واحداً، ونقرأ الصفحة المناسبة من القرص إلى الذاكرة، وننسخ العنصر المراد دفعه إلى الكلمة المناسبة في الصفحة، ونعيد كتابة الصفحة على القرص. أما عملية السحب، فهي عملية مشابهة. ننقص مؤشر المكس واحداً، ونقرأ الصفحة المناسبة من القرص، ونعيد قمة (أعلى) المكس. لا نحتاج إلى إعادة كتابة الصفحة لأنها لم تُعدّل.

لما كانت العمليات على القرص مكلفة نسبياً، فإننا نحسب تكلفتين لكل تنجيز: العدد الإجمالي لمرات النفاذ إلى القرص والزمن الإجمالي لوحدة للمعالجة المركزية. كل نفاذ إلى صفحة من  $m$  كلمة في القرص يكلف نفاذاً واحداً إلى القرص وزمناً  $\Theta(m)$  لوحدة للمعالجة المركزية.

أ. بالمقارنة، ما هو عدد مرات النفاذ إلى القرص في أسوأ الحالات في حالة  $\pi$  عملية على المكس باستخدام هذا التنجيز البسيط؟ ما هو الزمن الذي تستغرقه وحدة المعالجة المركزية في حالة  $\pi$  عملية على المكس؟ (عُرِّ عن الجواب بدلالة  $\pi$  و  $m$  في هذا الجزء والأجزاء التالية.)

لندرس الآن تنجيزاً للمكس نحتفظ فيه بصفحة من المكس في الذاكرة. (نحتفظ أيضاً بمقدار صغير من الذاكرة للاحتفاظ بأثر الصفحة الموجودة حالياً في الذاكرة.) لا يمكننا إجراء عملية على المكس إلا إذا كانت الصفحة المعنية من القرص في الذاكرة. عند الضرورة، يمكننا كتابة الصفحة الموجودة حالياً في الذاكرة إلى القرص وقراءة الصفحة الجديدة من القرص إلى الذاكرة. إذا كانت الصفحة المعنية من القرص موجودة سابقاً في الذاكرة، فهذا لا يتطلب نفاذاً إلى القرص.

ب. ما هو عدد مرات النفاذ إلى القرص في أسوأ الحالات التي تتطلبها  $\pi$  عملية دفع PUSH؟ ما هو الزمن الذي تستغرقه وحدة المعالجة المركزية؟



ت. ما هو عدد مرات النفاذ إلى القرص في أسوأ الحالات الذي تتطلبه  $n$  عملية على المكبس؟ ما هو الزمن الذي تستغرقه وحدة المعالجة المركزية؟

افترض أننا الآن ننجز للمكبس بالاحتفاظ بصفحتين في الذاكرة (إضافة إلى عدد صغير من الكلمات للاحتفاظ بمعلومات الحجز).

ث. صِف كيفية إدارة صفحات المكبس بحيث يكون العدد المستهلك لمرات النفاذ إلى القرص لأي عملية على المكبس  $O(1/m)$ ، والزمن المستهلك الذي تستغرقه وحدة المعالجة المركزية لأي عملية على المكبس  $O(1)$ .

### 2-18 الضم والتفريق في الأشجار 2-3-4

تأخذ عملية الضم  $join$  مجموعتين ديناميكيتين  $S'$  و  $S''$  وعنصرًا  $x$  بحيث يتحقق لكل عنصر  $x' \in S'$  و  $x'' \in S''$  للترتيبات:  $x'.key < x.key < x''.key$ . تعيد هذه العملية مجموعة  $S = S' \cup \{x\} \cup S''$ . تشبه عملية التفريق  $split$  عملية ضم "عكسي": إذا أُعطينا مجموعة ديناميكية  $S$  وعنصرًا  $x \in S$ ، نُنشئ عملية التفريق مجموعة  $S'$  تتألف من جميع عناصر  $S - \{x\}$  التي مفاتيحها أصغر من  $x.key$  ومجموعة  $S''$  تتألف من جميع عناصر  $S - \{x\}$  التي مفاتيحها أكبر من  $x.key$ . نتحرى في هذه المسألة، كيفية تنجيز هذه العمليات على الأشجار 2-3-4. نفترض للسهولة أن العناصر تتألف من مفاتيح فقط، وأن كل قيم المفاتيح متميزة.

أ. يَبين كيف يمكن الاحتفاظ بارتفاع الشجرة الفرعية ذات الجذر  $x$  كخاصة  $x.height$  لكل عقدة  $x$  من شجرة 2-3-4. تأكد أن تنجيزك لا يؤثر على الأزمنة المقاربة لتنفيذ البحث والإدراج والحذف.

ب. يَبين كيف يمكن تنجيز عملية الضم. في حالة شجرتين 2-3-4 هما  $T'$  و  $T''$  ومفتاح  $k$ ، يجب أن نُنفذ عملية الضم بزمن  $O(1 + |h' - h''|)$ ، حيث  $h'$  و  $h''$  هما ارتفاعا  $T'$  و  $T''$  على الترتيب.

ت. لندرس للمسار البسيط  $p$  من جذر شجرة 2-3-4 هي  $T$  إلى مفتاح ما  $k$ ، والمجموعة  $S'$  المكونة من مفاتيح  $T$  التي هي أصغر من  $k$ ، والمجموعة  $S''$  المكونة من مفاتيح  $T$  التي هي أكبر من  $k$ . يَبين أن  $p$  يقسم  $S'$  إلى مجموعة من الأشجار  $\{T'_0, T'_1, \dots, T'_m\}$  ومجموعة من المفاتيح  $\{k'_1, k'_2, \dots, k'_m\}$ ، حيث لكل قيم  $i = 1, 2, \dots, m$  يكون لدينا  $y < k'_i < z$  في حالة كل المفاتيح  $T'_{i-1}$  و  $T'_i$  و  $y \in T'_{i-1}$  و  $z \in T'_i$ . ما هي العلاقة بين ارتفاع الشجرتين  $T'_{i-1}$  و  $T'_i$ ؟ صِف كيف يقسم  $p$  المجموعة  $S''$  إلى مجموعتين من الأشجار والمفاتيح.

ث. يَبين كيف يمكن تنجيز عملية التفريق على  $T$ . استخدم عملية الضم لتجميع المفاتيح في  $S'$  ضمن شجرة

وحيدة 2-3-4 هي  $T'$  وتجميع المفاتيح في  $S'$  ضمن شجرة وحيدة 2-3-4 هي  $T''$ . يجب أن يكون زمن تنفيذ عملية التفريق  $O(\lg n)$ ، حيث  $n$  هو عدد المفاتيح في  $T$ . (لمصيح: يجب أن تكون تكاليف الضم مضمونة).

## ملاحظات الفصل

تعطي المراجع Knuth [211]، و Aho و Hopcroft و Ullman [5]، و Sedgewick [306] مناقشات إضافية عن أنواع الأشجار المتوازنة والأشجار المعشمة B-trees. يزود المرجع Comer [75] دراسة شاملة عن الأشجار المعشمة. ويناقش Sedgewick و Guibas [155] العلاقات بين الأنواع المختلفة للأشجار المتوازنة، ومنها الأشجار الحمراء-السوداء والأشجار 2-3-4.

في عام 1970 اخترع J. E. Hopcroft الأشجار 2-3، وهي نوع سابق للأشجار المعشمة والأشجار 2-3-4، وفيها لكل عقدة داخلية ابنان أو ثلاثة. أُدخل Bayer و McCreight [35] الأشجار المعشمة B-trees في عام 1972؛ ولم يشرحا سبب اختيارهما لهذا الاسم.

درس المرجع Bender و Demaine و Farach-Colton [40] كيفية جعل الأشجار المعشمة تعمل جيداً في وجود تأثيرات هرمية الذاكرة. تعمل خوارزميات *cache-oblivious* الخاصة بهم بفعالية عالية دون أن تعرف صراحةً حجم للمعطيات المنقولة ضمن هرمية الذاكرة.

تفيد بنية معطيات كومات فيوناتشي فائدة مضاعفة. الأولى هي أن هذه الكومات تدعم مجموعة من العمليات التي تولف ما يُعرف بالكومات "القابلة للدمج" "mergeable heap". والثانية هي أن العديد من عمليات كومات فيوناتشي يُنفَّذ بزمن محمَّد ثابت، وهو ما يجعل بنية المعطيات هذه مناسبة جدًا للتطبيقات التي تستدعي هذه العمليات بتواتر كبير.

### الكومات القابلة للدمج

**الكومة القابلة للدمج mergeable heap** هي أية بنية معطيات تدعم العمليات الخمس الآتية، لكل عنصر منها مفتاح  $key$ :

MAKE-HEAP() تُنشئ وتعيد كومة جديدة لا تحتوي أي عنصر.

INSERT( $H, x$ ) تُدرج عنصرًا  $x$  - مُلئ مفتاحه سابقًا - ضمن كومة  $H$ .

MINIMUM( $H$ ) تُعيد مؤشرًا إلى العنصر الذي يكون مفتاحه أصغرًا ضمن  $H$ .

EXTRACT-MIN( $H$ ) تحذف العنصر ذا المفتاح الأصغر من  $H$  وتُعيد مؤشرًا إليه.

UNION( $H_1, H_2$ ) تُنشئ كومة جديدة تتضمن جميع عناصر الكومتين  $H_1$  و  $H_2$  وتعيدها. يجري في هذه العملية "تدمير" الكومتين  $H_1$  و  $H_2$ .

إضافة إلى عمليات الكومات القابلة للدمج المذكورة، تدعم كومات فيوناتشي أيضًا العمليتين الآتيتين:

DECREASE-KEY( $H, x, k$ ) تُسند إلى العنصر  $x$  ضمن الكومة  $H$  قيمة جديدة للمفتاح  $k$ ، يُفترض ألا تكون أكبر من قيمة مفتاحها الحالي.<sup>1</sup>

<sup>1</sup> كما أشرنا في مقدمة الباب الخامس، الكومات المفترضة القابلة للدمج هي الكومات الأصغرية القابلة للدمج mergeable min-heaps، ومن ثم فإنَّ العمليات MINIMUM و EXTRACT-MIN و DECREASE-KEY قابلة للتطبيق. يمكننا بالمقابل تعريف **كومة أعظمية قابلة للدمج mergeable max-heap** مع العمليات MAXIMUM و EXTRACT-MAX و INCREASE-KEY.

$DELETE(H, x)$  يحذف العنصر  $x$  من الكومة  $H$ .

يُدرج الجدول في الشكل 1.19 أنه إذا لم تكن ثمة حاجة إلى العملية UNION، فإن الكومات الثنائية المعتادة - كالمستخدمة في الفرز بالكومة (الفصل 6) - تعمل جيدًا تقريبًا. تُنفَّذ العمليات الأخرى في أسوأ الحالات بزمن  $O(\lg n)$  على كومة ثنائية. لكن إذا كنا نحتاج إلى دعم العملية UNION، فإن أداء الكومات الثنائية يكون ضعيفًا. عند ضم الصفيفتين اللتين تحملان الكومتين الثابنتين المراد دمجهما ثم تنفيذ الإجراء BUILD-MIN-HEAP (انظر للمقطع 3.6)، فإن العملية UNION تستغرق زمنًا  $\Theta(n)$  في أسوأ الحالات.

وبالمقابل، فإن كومات فيبوناتشي لها حدود أفضل لزمن التنفيذ للمقارب من الكومات الثنائية للعمليات: INSERT، و UNION، و DECREASE-KEY ولها أزمته التنفيذ للمقارب نفسها لبقية العمليات. وتجدر ملاحظة أن أزمته التنفيذ لكومات فيبوناتشي في الشكل 1.19 هي حدود أزمته مُحَدَّدة، وليست حدود أزمته في أسوأ الحالات لكل عملية. نأخذ عملية UNION زمنًا مُحَدَّدًا ثابتًا في كومات فيبوناتشي أفضل بكثير من زمن الحالة الأسوأ الخطي الذي تتطلبه الكومات الثنائية (طبعًا بافتراض أن الحد الزمني المُحَدَّد كافٍ).

### كومات فيبوناتشي من الناحية النظرية والعملية

من الناحية النظرية، كومات فيبوناتشي مرغوبة بوجه خاص عندما يكون عدد عمليات EXTRACT-MIN و DELETE قليلًا بالنسبة إلى عدد العمليات الأخرى للنسرة. وهذه الحالة تظهر في عدة تطبيقات. فعلى سبيل المثال، قد تستدعي بعضُ خوارزميات مسائل البيان (graph problems) العملية DECREASE-KEY مرة لكل وصلة edge. في البيانات الكثيفة التي تتضمن وصلات كثيرة، يقدم الزمن للمُحَدَّد  $\Theta(1)$

الإجراء	كومة ثنائية (أسوأ الحالات)	كومة فيبوناتشي (مُحَدَّدة)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

**الشكل 1.19** أزمته تنفيذ العمليات على تنجيزي الكومات القابلة للدمج. نمرز لعدد العناصر في الكومة (الكومات) عند تطبيق كل عملية  $n$ .

لاستدعاء DECREASE-KEY تحسبًا كبيرًا على زمن أسوأ الحالات  $\Theta(\lg n)$  في الكومات الثنائية. تعتمد الخوارزميات السريعة لمسائل مثل حساب أشجار المسح الصغرى (الفصل 23) وإيجاد أقصر المسارات من منبع وحيد (الفصل 24) في أساسها على كومات فيوناتشي.

أما من الناحية العملية، فإن العوامل الثابتة وتعقيد البرجة تقلل من الرغبة في استخدام كومات فيوناتشي نسبةً إلى الكومات الثنائية العادية (أو الكومات من الدرجة  $K$ ) في معظم التطبيقات، باستثناء بعض التطبيقات التي تدير حجمًا كبيرًا من المعطيات. لذلك فإن كومات فيوناتشي لها غالبًا أهمية نظرية. ولكن إذا جرى تطوير بنية معطيات بسيطة لها الحدود الزمنية المحسّنة لكومات فيوناتشي نفسها، فيكون لها استخدام عملي أيضًا.

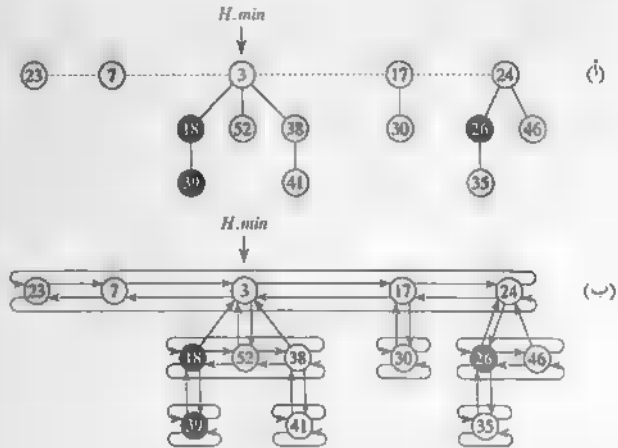
لا تدعم الكومات الثنائية وكومات فيوناتشي عملية البحث SEARCH على نحو فعال؛ لأن العنود على عقدة تحتوي مفتاحًا معينًا يمكن أن يأخذ وقتًا طويلاً. ولهذا السبب، تتطلب العمليات مثل DECREASE-KEY و DELETE المطبقة على عنصرٍ معين مؤشرًا إلى ذلك العنصر باعتبارها جزءًا من دخلها. وكما في مناقشتنا لأرنال الأفضليات في المقطع 5.6، عندما نستخدم كومة قابلة للدمج في تطبيق ما، نُخزّن غالبًا مقبضًا *handle* لفرض التطبيق الموافق في كل عنصر من الكومة القابلة للدمج، وكذلك مقبضًا للعنصر الموافق من الكومة القابلة للدمج لكل غرض من أغراض التطبيق. يعتمد تحديد طبيعة هذه المقابض بدقة على التطبيق وتجهيزه.

تعتمد كومات فيوناتشي، كما رأينا في عدة بنى معطيات أخرى، على الأشجار ذات الجذور. نُقل كل عنصر بعقدة ضمن شجرة، وكل عقدة لها واصفة *key*. سنستخدم التعبير "عقدة" بدلًا من "عنصر" فيما تبقى من هذا الفصل. كما أننا ستحتاج أمور تخصيص العقد قبل الإدراج وتحريرها بعد الحذف، حيث نفترض أن الرمز الذي يستدعي إجراءات الكومات يعالج هذه التفاصيل.

يُعرف المقطع 1.19 كومات فيوناتشي، وناقش كيفية تمثيلها، وبُذمت دالة الكمون المستخدمة في تحليلها المخدّد. ويُظهر المقطع 2.19 كيفية تحجيز عمليات الكومات القابلة للدمج وكيفية الوصول إلى الحدود الزمنية المخدّدة الظاهرة في الشكل 1.19. يجري عرض العاملين المتبقيين DECREASE-KEY و DELETE في المقطع 3.19. وأخيرًا، ينتهي للمقطع 4.19 جزءًا هامًا من التحليل ويوفر هذا الاسم الغريب لبنية المعطيات.

## 1.19 بنية كومات فيوناتشي

**كومة فيوناتشي** *Fibonacci heap* هي تجمّع من الأشجار ذات الجذر المرتبة بترتيب الكومات وفق الأصغر *min-heap ordered*. أي إن كل شجرة تخضع لخاصة الكومات وفق الأصغر *min-heap property*: مفتاح عقدة ما هو أكبر أو يساوي مفتاح أيها. يُظهر الشكل 2.19 (أ) مثالاً على كومة فيوناتشي.



**الشكل 2.19** (أ) كومة فيوناتشي مؤلفة من خمسة أشجار مرتبة بترتيب الكومات وفق الأصغر و 14 عقدة. يشرح الخط المنقطع إلى لائحة الجذور. العقدة الصفراء في الكومة هي العقدة التي تحتوي المفتاح 3. جرى تلوين العقد المعلمة باللون الأسود. كمن هذه الكومة الخاصة هو  $5 + 2 \cdot 3 = 11$ . (ب) تمثيل أكثر اكتمالاً يظهر مؤشرات  $p$  (أسهم صاعدة) و  $child$  (أسهم هابطة)، و  $left$  و  $right$  (أسهم جانبية). تُعَيَّن بقية الأشكال في هذا الفصل هذه التفاصيل، لأنَّ بالإمكان تحديد كل المعلومات للمروضة هنا مما يظهر في الجزء (أ).

وكما يُظهر الشكل 2.19(ب)، تتضمن كل عقدة  $x$  مؤشرًا إلى أبيها  $x.p$  ومؤشرًا إلى أحد أبنائها  $x.child$ . يربط أبناء  $x$  مع بعضهم بلائحة دائرية مضاعفة الترابط، نسميها **لائحة أبناء**  $x.child\ list$ . يتضمن كل ابن  $y$  من لائحة الأبناء مؤشرين إلى أعوي  $y$  الأيسر والأيمن  $y.left$  و  $y.right$  على الترتيب. فإذا كانت العقدة  $y$  هي الابن الوحيد، يكون  $y.left = y.right = y$ . يمكن أن يظهر الأشقاء في لائحة الأبناء بأي ترتيب.

إنَّ لاستخدام اللوائح الدائرية المضاعفة الترابط (انظر للمقطع 2.10) في كومات فيوناتشي فالدتئين. أولاً، يمكننا إدراج عقدة في أي مكان أو حذف عقدة من أي مكان في لائحة دائرية مضاعفة الترابط بزم  $O(1)$ . ثانياً، إذا كان لدينا لائحتين من هذا النمط، يمكننا ضمهما (أو وصلهما معاً) في لائحة دائرية مضاعفة الترابط بزم  $O(1)$ . سنتطرق إلى هذه العمليات شكلياً عند وصف العمليات على كومات فيوناتشي، تاركين للقارئ ملء تفاصيل تنجزها إن رغب بذلك.

تتضمن كل عقدة واصفتان آخران. تخزّن عدد الأبناء في لائحة الأبناء في عقدة  $x$  وهو عزن  $x.degree$ . والواصفة ذو القيمة للمنطقة  $x.mark$  يشير إلى فقدان العقدة  $x$  ابن لها منذ آخر مرة أصبحت فيها ابناً لعقدة أخرى. تكون العقد المنشأة حديثاً غير معلّمة، وتصبح أية عقدة  $x$  غير معلّمة

عندما تكون ابناً لعقدة أخرى. سنكتفي بوضع قيمة FALSE في جميع الواصفات *mark*، إلى أن نتطرق إلى العملية DECREASE-KEY في المقطع 3.19.

يجري النفاذ إلى كومة فيوناتشي  $H$  عن طريق مؤشر  $H.min$  إلى جذر شجرة تتضمن مفتاحاً أصغر (تسمى هذه العقدة **العقدة الصغرى** *minimum node* لكومة فيوناتشي. إذا كان هناك أكثر من جذر له مفتاح بالقيمة الصغرى نفسها، فيمكن أن نعتبر أبناً منها العقدة الصغرى. إذا كانت كومة فيوناتشي  $H$  فارغة، فإن  $H.min = NIL$ .

ترتبط جذور جميع الأشجار في كومة فيوناتشي ببعضها ببعض باستخدام مؤشرات *left* و *right* بلائحة دائرية مضاعفة الترابط نسميها **لائحة جذور** *root list* كومة فيوناتشي. يشير المؤشر  $H.min$  إذن إلى العقدة ذات المفتاح الأصغر في لائحة الجذور. يمكن أن تظهر الأشجار في لائحة الجذور بأي ترتيب. نعلم على وصفة أخرى في كومة فيوناتشي  $H$  هي  $H.n$ : العدد الحالي للعقد في  $H$ .

### دالة الكمون

سنستخدم كما ذكرنا طريقة الكمون المعروضة في المقطع 3.17 لتحليل أداء العمليات على كومات فيوناتشي. في حالة كومة فيوناتشي معطاة  $H$ ، يكون  $\tau(H)$  عدد الأشجار في لائحة جذور  $H$  و  $m(H)$  عدد العقد المعلّمة في  $H$ . فنعرّف كمون كومة فيوناتشي  $H$  كالآتي:

$$\Phi(H) = \tau(H) + 2m(H) \quad (1.19)$$

(سنعرف المزيد عن دالة الكمون في المقطع 3.19). فمثلاً، إن كمون كومة فيوناتشي الظاهرة في الشكل 2.19 هو  $11 = 5 + 2 \cdot 3$ . وإن كمون مجموعة من كومات فيوناتشي هو مجموع كمونات كومات فيوناتشي المؤلفة لها. سنفترض أن وحدة الكمون يمكن أن تسدد ثمن مقدار ثابت من العمل كبير كفاية ليسدد تكلفة أية أجزاء عمل محددة نابعة الزمن قد نواجهها.

نفترض أن تطبيق كومة فيوناتشي يبدأ بدون كومات. فيكون الكمون الأولي 0، وبحسب المعادلة 1.19، لا يمكن أن يكون الكمون سالباً في المرات التالية. وبحسب المعادلة 3.17، يكون الحد الأعلى للتكلفة الكلية المتحددة هو حداً أعلى للتكلفة الكلية الفعلية لمتتالية العمليات.

### الدرجة العظمى

نفترض التحليلات المتحددة التي سنجرها في المقاطع المتبقية من هذا الفصل أن هناك حداً أعلى معروفاً  $D(n)$  للدرجة العظمى لأيّة عقدة في كومة فيوناتشي من  $n$  عقدة. لن نُثبت ذلك، إلا عندما تُدعم عمليات الكومات القابلة للدمج فقط،  $D(n) \leq \lg n$ . (يُطلب إليك في المسألة 2-19 (ث) إثبات هذه الخاصية.) سنرى في المقطع 3.19 و 4.19 أنه عندما ندعم DECREASE-KEY و DELETE أيضاً يكون  $D(n) = O(\lg n)$ .

## 2.19 عمليات الكومات القابلة للدمج

تؤخر عمليات الكومات القابلة للدمج على كومات فيوناتشي العمل قدر المستطاع. فهناك تسوية (مقايضة) للأداء بين تنحيات العمليات المختلفة. فإذا أدرجنا مثلاً عقدة بإضافتها إلى لائحة الجذور، فإن هذا يتطلب زمناً ثابتاً فقط. أما إذا كنا قد بدأنا من كومة فيوناتشي فارغة، ثم أدرجنا  $k$  عقدة، فستتألف الكومة من لائحة جذور ذات  $k$  عقدة فقط. وتكون التسوية (المقايضة) بأننا إذا نُقِّدنا عملية EXTRACT-MIN على كومة فيوناتشي  $H$ ، بعد حذف العقدة التي يشير عليها  $H.min$ ، فعلينا أن ننظر ضمن كل عقدة من العقد  $k-1$  المتبقية في لائحة الجذور للعثور على العقدة الصغرى الجديدة. وأثناء مرورنا على كامل لائحة الجذور خلال عملية EXTRACT-MIN فإننا نقوم أيضاً بتدعيم consolidate العقد وتحويلها إلى أشجار كومات مرتبة وفق الأصغر لتقليص حجم لائحة الجذور. سنرى أنه لا يهم شكل لائحة الجذور قبل عملية EXTRACT-MIN، فبعد ذلك سيكون لكل عقدة في لائحة الجذور درجة واحدة ضمن لائحة الجذور، وهذا يؤدي إلى لائحة جذور حجمها  $D(n) + 1$ .

## إنشاء كومة فيوناتشي جديدة

لإنشاء كومة فيوناتشي فارغة يُخصَّص الإجراء MAKE-FIB-HEAP غرض كومة فيوناتشي  $H$  وبعبارة، حيث  $H.n = 0$  و  $H.min = NIL$ ؛ أي ليس هناك أشجار في  $H$ . ولما كان  $t(H) = 0$  و  $m(H) = 0$  فإن كمون كومة فيوناتشي الفارغة هو  $\Phi(H) = \square$ . وبذلك تساوي التكلفة المخطئة للإجراء MAKE-FIB-HEAP تكلفته الفعلية  $O(1)$ .

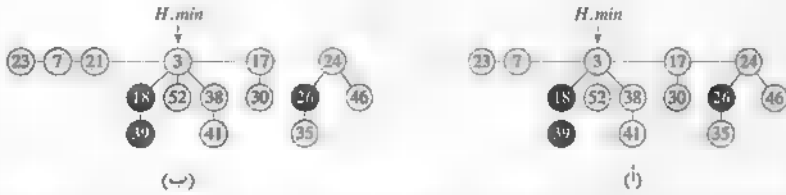
## إدراج عقدة

يقوم الإجراء التالي بإدراج عقدة  $x$  في كومة فيوناتشي  $H$ ، مفترضاً أن العقدة جرى فحصتها وأن المفتاح  $x.key$  جرى ملؤه سلفاً.

FIB-HEAP-INSERT( $H, x$ )

- 1  $x.degree = 0$
- 2  $x.p = NIL$
- 3  $x.child = NIL$
- 4  $x.mark = FALSE$
- 5 **if**  $H.min == NIL$
- 6     create a root list for  $H$  containing just  $x$
- 7      $H.min = x$
- 8 **else** insert  $x$  into  $H$ 's root list
- 9     **if**  $x.key < H.min.key$
- 10          $H.min = x$
- 11  $H.n = H.n + 1$





**الشكل 3.19** إدراج عقدة في كومة فيوناتشي. (أ) كومة فيوناتشي  $H$ . (ب) كومة فيوناتشي  $H$  بعد إدراج العقدة ذات المفتاح 21. تصبح العقدة شجرة مرتبة بترتيب الكومة وفق الأصغر ثم تُضاف إلى لائحة الجذور لتصبح الأخ الأيسر للعقدة الصغرى.

تقوم الأسطر 1-4 باستبداء الواصفات البنيوية للعقدة  $x$ . يختار السطر 5 كون كومة فيوناتشي  $H$  فارغة؛ فإذا كانت كذلك يجعل السطران 6-7 العقدة  $x$  العقدة الوحيدة في لائحة جذور  $H$  ويجهلان  $H.min$  تشير إلى  $x$ . وإلا، فإن الأسطر 8-10 تدرج  $x$  ضمن لائحة جذور  $H$  تُحدّث  $H.min$  إذا كان ذلك ضرورياً. أخيراً، يزيد السطر 10 قيمة  $H.n$  ليأخذ بذلك إضافة العقدة الجديدة بالاعتبار. يُظهر الشكل 3.19 عقدة مفتاحها 21 مدرجة في كومة فيوناتشي الظاهرة في الشكل 2.19.

ولتحديد التكلفة المخمّدة للإجراء FIB-HEAP-INSERT، نفترض أن  $H$  كومة فيوناتشي المدخلة و  $H'$  كومة فيوناتشي الناتجة. عندها يكون لدينا  $t(H') = t(H) + 1$  و  $m(H') = m(H)$ ، ونكون الزيادة في الكومن هي:

$$((t(H) + 1) + 2m(H)) - (t(H) + 2m(H)) = 1.$$

ولما كانت التكلفة الفعلية هي  $O(1)$ ، فإن التكلفة المخمّدة  $O(1) + 1 = O(1)$ .

### إيجاد العقدة الصغرى

تُعطى العقدة الصغرى في كومة فيوناتشي  $H$  بال مؤشر  $H.min$ ، ومن ثم يمكننا إيجاد العقدة الصغرى برمز فعلي  $O(1)$ . ولما كان كومن  $H$  لا يتغير، فإن التكلفة المخمّدة لهذه العملية تساوي تكلفتها الفعلية  $O(1)$ .

### توحيد كومتين فيوناتشي

يؤخّذ الإجراء التالي كومتين فيوناتشي  $H_1$  و  $H_2$  مع تدمير الكومتين الأصليتين خلال الإجراءية. وهو يقوم ببساطة بضم لائحتي جذور  $H_1$  و  $H_2$  ثم يجد العقدة الصغرى الجديدة. بعد ذلك لن يجري استخدام الأغراض التي تمثل  $H_1$  و  $H_2$ .

```
FIB-HEAP-UNION( $H_1, H_2$ )
1   $H = \text{MAKE-FIB-HEAP}()$ 
2   $H.min = H_1.min$ 
```

- 3 concatenate the root list of  $H_2$  with the root list of  $H$
- 4 if  $(H_1.min == NIL)$  or  $(H_2.min \neq NIL \text{ and } H_2.min.key < H_1.min.key)$
- 5      $H.min = H_2.min$
- 6  $H.n = H_1.n + H_2.n$
- 7 return  $H$

نضم الأسطر 1-3 لاحتجى جذور  $H_1$  و  $H_2$  في لائحة جذور جديدة  $H$ . نحدد الأسطر 2 و 4 و 5 العقدة الصغرى في  $H$ ، ويضع السطر 6 العدد الكلي للعقد في  $H.n$ . يعيد السطر 7 كومة فييوناتشي الناتجة  $H$ . وكما في إجراء FIB-HEAP-INSERT تبقى الجذور كلها جذورًا.

التغير في الكمون هو

$$\begin{aligned} \Phi(H) &= (\Phi(H_1) + \Phi(H_2)) \\ &= (t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2))) \\ &= 0, \end{aligned}$$

لأن  $t(H) = t(H_1) + t(H_2)$  و  $m(H) = m(H_1) + m(H_2)$ . وبذلك، تكون التكلفة المحسنة للإجراء FIB-HEAP-UNION مساويةً تكلفته الفعلية  $O(1)$ .

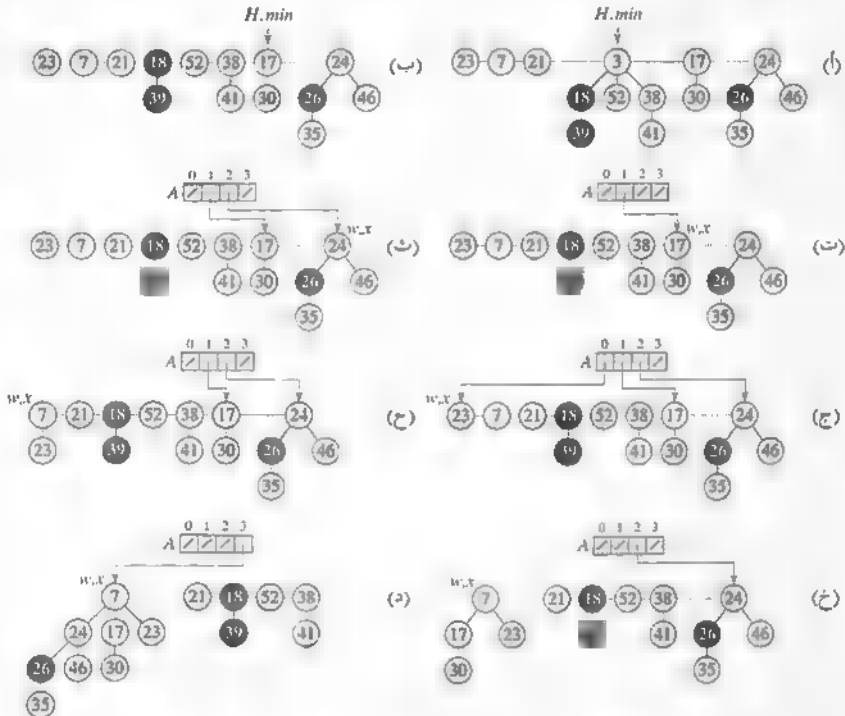
### استخراج العقدة الصغرى

تعتبر إجرائية استخراج العقدة الصغرى أعقد العمليات المعروضة في هذا المقطع. وهي أيضًا العملية التي يُجرى فيها العمل المألوف لتدعيم الأشجار في لائحة الجذور. يُستخرج شبه الرماز التالي للعقدة الصغرى. يفترض الرماز اصطلاحًا أنَّ المؤشرات المتبقية في اللائحة للترابطة تُحدَّث عند حذف عقدة من اللائحة للترابطة، لكن تبقى المؤشرات في العقدة المستخرجة دون تغيير. يُستخدم الرماز أيضًا إجراءً مساعدًا CONSOLIDATE سنراه باختصار.

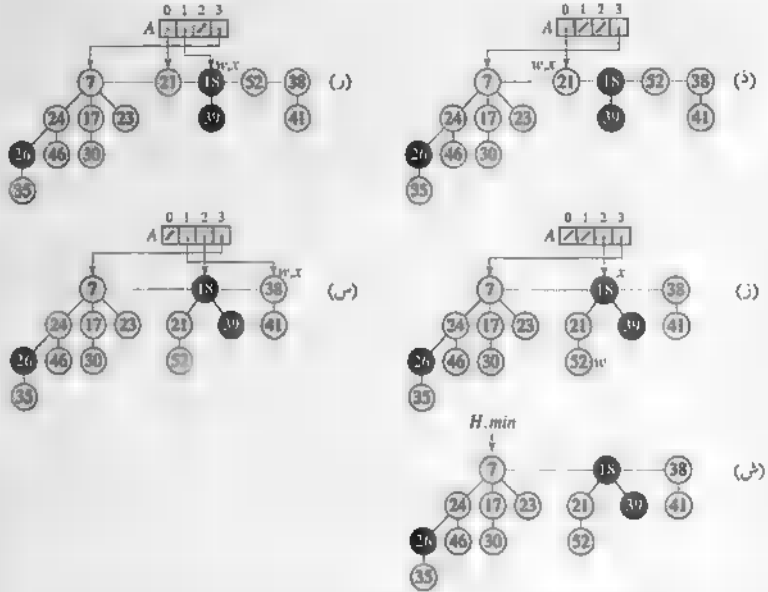
#### FIB-HEAP-EXTRACT-MIN( $H$ )

- 1  $z = H.min$
- 2 if  $z \neq NIL$
- 3     for each child  $x$  of  $z$
- 4         add  $x$  to the root list of  $H$
- 5          $x.p = NIL$
- 6     remove  $z$  from the root list of  $H$
- 7     if  $z == z.right$
- 8          $H.min = NIL$
- 9     else  $H.min = z.right$
- 10     CONSOLIDATE( $H$ )
- 11      $H.n = H.n - 1$
- 12 return  $z$

وكما يوضح الشكل 4-19، يقوم FIB-HEAP-EXTRACT-MIN أولاً بجعل كل عقدة من أبناء العقدة الصغرى جذراً وحذف العقدة الصغرى من لائحة الجذور. ثم يدعم لائحة الجذور بربط الجذور ذات الدرجات المتساوية إلى أن يبقى على الأكثر جذر واحد من كل درجة.



**الشكل 4-19 عمل FIB-HEAP-EXTRACT-MIN.** (أ) كومة فيبوناتشي  $H$ . (ب) الوضع بعد حذف العقدة الصغرى  $z$  من لائحة الجذور وإضافة أبنائها إلى هذه اللائحة. (ت)-(ج) الصيغة  $A$  والأشجار بعد كل من التكرارات الثلاثة الأولى لحلقة **for** في الأسطر 14-4 من الإجراء CONSOLIDATE. يعالج الإجراء لائحة الجذور بدءاً من الجذر الذي يشير إليه  $H.min$  وباتجاه المؤشرات اليمينية  $right$ . يظهر كل جزء قيم  $w$  و  $x$  في نهاية تكرار ما. (د)-(هـ) التكرار التالي للحلقة **for** مع إظهار قيم  $w$  و  $x$  في نهاية كل تكرار لحلقة **while** في الأسطر 7-13. يظهر الجزء (ج) الوضع بعد المرور الأول في الحلقة **while**. جرى ربط العقدة ذات المفتاح 23 بالعقدة ذات المفتاح 7، والتي يشير إليها  $x$  الآن. في الجزء (د)، جرى ربط العقدة ذات المفتاح 17 بالعقدة ذات المفتاح 7 التي مازال  $x$  يشير إليها. في الجزء (هـ)، جرى ربط العقدة ذات المفتاح 24 بالعقدة ذات المفتاح 7. ولما كانت  $A[3]$  لا تشير سلفاً إلى أية عقدة في نهاية تكرار حلقة **for**، فيجري وضع المؤشر  $A[3]$  على جذر الشجرة الناتجة.



يُتَّبَع الشكل 4.19 (ذ)-(س) الوضع بعد كل من التكرارات الأربعة التالية للحلقة for. (ش) كومة فيوناتشي  $H$  بعد إعادة بناء لائحة الجذور من الصفيفة  $A$  وتحديد المؤشر الجديد  $H.min$ .

نبدأ من السطر 1 بتعزيين مؤشر  $z$  إلى العقدة الصغرى؛ يُعيد الإجراء هذا المؤشر في النهاية. إذا كان  $z = NIL$ ، فإن كومة فيوناتشي  $H$  هي فارغة أصلاً، ونكون قد انتهينا. وإلا نُحذف العقدة  $z$  من  $H$ ، وذلك بعمل جميع أبناء  $z$  جنوداً في  $H$  في الأسطر 3-5 (يوضعهم في لائحة الجذور)، وحذف  $z$  من لائحة الجذور في السطر 6. إذا كان  $z$  هو نفسه الأخ الأيمن بعد السطر 6، تكون  $z$  هي العقدة الوحيدة في لائحة الجذور ولا يكون لديها أبناء، فكل ما تبقى هو إفراغ كومة فيوناتشي في السطر 7 قبل إعادة  $z$ . إن لم يكن كذلك، نضع المؤشر  $H.min$  على لائحة الجذور ليُشير إلى جذر مختلف عن  $z$  (في هذه الحالة الأخ الأيمن له)، والتي قد لا تكون بالضرورة العقدة الصغرى الجديدة عند انتهاء FIB-HEAP-EXTRACT-MIN. يظهر الشكل 4.19(ب) كومة فيوناتشي المعروضة في الشكل 4.19(أ) بعد تنفيذ السطر 9.

الخطوة التالية التي نقلص فيها عدد الأشجار في كومة فيوناتشي هي تدعيم *consolidating* لائحة جذور  $H$  التي يقوم بها طلب CONSOLIDATE( $H$ ). يكون تدعيم لائحة الجذور بتنفيذ متكرر للخطوات التالية إلى أن يصبح لكل جذر في لائحة الجذور درجة *degree* متمايزة.

1. اكتشف جذرين  $x$  و  $y$  لهما الدرجة نفسها في لائحة الجذور. ودون فقدان العمومية، ليكن  $x.key \leq y.key$ .

2. اربط  $y$  بـ  $x$ : احذف  $y$  من لائحة الجذور، واجعل  $y$  ابناً لـ  $x$  بطلب الإجراء FIB-HEAP-LINK. يزيد هذا الإجراء قيمة الوصفة  $x.degree$  ويزيل العلامة من  $y$ .

يستخدم الإجراء CONSOLIDATE صفيقة مساعدة  $A[0..D(H,n)]$  لتتبع الجذور وفقاً لدرجاتها. إذا كان  $A[i] = y$ ، فيكون  $y$  حاليًا جذراً يحقق  $y.degree = i$ . بهدف تخصيص الصفيقة يجب أن نعرف طبقاً كيفية حساب الحد الأعلى  $D(H,n)$  للدرجة العظمى، لكننا سنرى كيف نفعل ذلك في المقطع 4.19.

CONSOLIDATE( $H$ )

```

1  let  $A[0..D(H,n)]$  be a new array
2  for  $i = 0$  to  $D(H,n)$ 
3       $A[i] = \text{NIL}$ 
4  for each node  $w$  in the root list of  $H$ 
5       $x = w$ 
6       $d = x.degree$ 
7      while  $A[d] \neq \text{NIL}$ 
8           $y = A[d]$  // another node with the same degree as  $x$ .
9          if  $x.key > y.key$ 
10             exchange  $x$  with  $y$ 
11             FIB-HEAP-LINK( $H, y, x$ )
12              $A[d] = \text{NIL}$ 
13              $d = d + 1$ 
14              $A[d] = x$ 
15   $H.min = \text{NIL}$ 
16  for  $i = 0$  to  $D(H,n)$ 
17      if  $A[i] \neq \text{NIL}$ 
18          if  $H.min == \text{NIL}$ 
19             create a root list for  $H$  containing just  $A[i]$ 
20              $H.min = A[i]$ 
21          else insert  $A[i]$  into  $H$ 's root list
22          if  $A[i].key < H.min.key$ 
23              $H.min = A[i]$ 
```

FIB-HEAP-LINK( $H, y, x$ )

```

1  remove  $y$  from the root list of  $H$ 
2  make  $y$  a child of  $x$ , incrementing  $x.degree$ 
3   $y.mark = \text{FALSE}$ 
```

يعمل الإجراء CONSOLIDATE تفصيليًا كالآتي. نخصّص الأسطر 1-3 الصفيقة A وتجعل قيم كل عنصر فيها NIL. تعالج حلقة for في الأسطر 4-14 كل جذر w في لائحة الجذور. أثناء ربط الجذور، يمكن ربط w بعقدة أخرى، ومن ثم لا تبقى جذورًا. ومع ذلك، تبقى  $\blacksquare$  دوماً ضمن شجرة لها جذر ما x الذي قد يكون هو w نفسه أو لا. ولما كنا نريد جذورًا واحدًا على الأكثر من كل درجة، فإننا ننظر إلى الصفيقة A لنرى: هل تحتوي على جذر y له درجة x نفسها؟ فإذا كانت كذلك نربط الجذرين x و y، ولكن مع ضمان بقاء x جذرًا بعد الربط. أي نربط y بـ x بعد تبديل مؤشرات الجذرين إذا كان مفتاح y أصغر من مفتاح x. بعد ربط y بـ x، نزيد درجة x بمقدار 1، وهكذا نتابع هذه الإجرائية، نربط x بجذر آخر تتساوى درجته مع درجة x الجديدة، إلى أن لا يبقى هناك جذر من الجذور التي عالجناها له درجة x نفسها. ثم نجعل عنصر A للموافق يشير إلى x بحيث نكون قد سجلنا أن x هو الجذر الوحيد من درجته الذي عالجناه سلفًا عندما نعالج جذورًا أخرى فيما بعد. عندما تنتهي حلقة for هذه، سيبقى على الأكثر جذر واحد من كل درجة، وستشير الصفيقة A إلى كل جذر متبقي.

تكرر حلقة while في الأسطر 7-13 ربط الجذر x للشجرة التي تحتوي العقدة w بشجرة أخرى بلجذرها درجة x نفسها، وذلك إلى أن لا يكون لجذر آخر الدرجة نفسها. نحافظ حلقة while هذه على اللامتغير التالي:

في بداية كل تكرار للحلقة while يكون  $d = x.degree$ .

نستخدم لامتغير الحلقة هذا كالآتي:

الاستبداء: يضمن السطر 6 تحقق لامتغير الحلقة أول مرة ندخل فيها الحلقة.

المحافظة: في كل تكرار لحلقة while، يشير  $A[d]$  إلى جذر ما y. لَمَّا كان  $d = x.degree = y.degree$  فإننا سنربط x بـ y. ومن يملك منهما المفتاح الأصغر يكون أبًا للآخر بعد عملية الربط، ولذلك يبدّل السطران 9-10 المؤشرين إلى x و y إذا كان ذلك ضروريًا. ثم نربط y بـ x باستدعاء  $FIB-HEAP-LINK(H, y, x)$  في السطر 11. نزيد هذا الاستدعاء من قيمة  $x.degree$  لكنه يترك  $y.degree$  كما هي d. وحيث إن العقدة y لم تُعد جذرًا، لذا فإن السطر 12 يُحذف المؤشر إليها من الصفيقة A. ولما كان استدعاء  $FIB-HEAP-LINK$  يزيد قيمة  $x.degree$ ، فإن السطر 13 يستعيد اللامتغير  $d = x.degree$ .

الانتهاء: نكرر الحلقة while إلى أن يصبح  $A[d] = NIL$ ، وفي هذه الحالة لا يكون هناك جذر آخر له درجة x نفسها.

بعد انتهاء حلقة while نجعل  $A[d]$  يشير إلى x في السطر 14 ونقوم بالتكرار التالي لحلقة for.

تُظهر الأشكال 4.19 (ت) -- (ج) الصيغة  $A$  والأشجار الناتجة بعد التكرارات الثلاثة الأولى لحلقة **for** في الأسطر 14-4. في التكرار التالي لحلقة **for**، تحصل ثلاث عمليات ربط؛ تُظهر نتائجها في الأشكال 4.19 (ح) -- (د). تُظهر الأشكال 4.19 (ذ) -- (س) نتيجة التكرارات الأربعة التالية لحلقة **for**. كل ما تبقى هو التنظيف. عندما تنتهي حلقة **for** في الأسطر 14-4، يُفرغ السطر 15 لائحة الجذور، وتعيد الأسطر 16-23 بناءها اعتبارًا من الصيغة  $A$ . تُظهر كومة فيوناتشي الناتجة في الشكل 4.19 (ش). بعد تدعيم لائحة الجذور يُهيئ **FIB-HEAP-EXTRACT-MIN** عمله بإنقاص قيمة  $H.n$  في السطر 11 وإعادة مؤشر إلى العقدة المحذوفة  $z$  في السطر 12.

نبين الآن أنَّ التكلفة المخمّدة لاستخراج العقدة الصغرى من كومة فيوناتشي ذات  $n$  عقدة هي  $O(D(n))$ . نرمز بـ  $H$  لكومة فيوناتشي قبل عملية **FIB-HEAP-EXTRACT-MIN** مباشرة.

نبدأ بحساب الكلفة الفعلية لاستخراج العقدة الصغرى. تأتي مساهمة الحد  $O(D(n))$  من كون **FIB-HEAP-EXTRACT-MIN** يعالج  $D(n)$  أبنا على الأكثر من أبناء العقدة الصغرى، ومن العمل في الأسطر 2-3 و 16-23 من **CONSOLIDATE**. يبقى تحليل مساهمة الحلقة **for** في الأسطر 4-14 من **CONSOLIDATE**، والتي نستخدم لأجلها تحليلًا مجتمعا. إنَّ حجم لائحة الجذور عند طلب **CONSOLIDATE** هو  $t(H) + D(n)$  على الأكثر، لأنها تتألف من عقد لائحة الجذور الأصلية  $t(H)$  مطروحا منها عقدة الجذر المستخرجة، ومضافا إليها أبناء العقدة المستخرجة الذين لا يتعدى عددهم  $D(n)$ . ضمن تكرار معين من حلقة **for** في الأسطر 14-4 يعتمد عدد تكرارات الحلقة **while** في الأسطر 7-13 على لائحة الجذور. لكننا نعلم أنَّه في كل مرور في الحلقة **while** يجرى ربط أحد الجذور بجذر آخر، وبذلك يكون العدد الكلي لتكرارات حلقة **while** مع كل تكرارات حلقة **for** هو على الأكثر عدد الجذور في لائحة الجذور. ومن ثمَّ يتناسب مقدار العمل الكلي المنفَّذ في حلقة **for** على الأكثر مع  $t(H) + D(n)$ . فيكون العمل الفعلي الكلي لاستخراج العقدة الصغرى هو  $O(D(n) + t(H))$ .

إنَّ الكومون قبل استخراج العقدة الصغرى هو  $t(H) + 2m(H)$ ، والكومون بعد ذلك هو  $t(H) + 2m(H) + D(n) + 1$  على الأكثر، لأن لدينا  $D(n) + 1$  جذرًا متبقيًا على الأكثر، ولم يجر تعليم أية عقدة خلال العملية. فتكون التكلفة للمخمّدة على الأكثر

$$\begin{aligned} & O(D(n) + t(H)) + ((D(n) + 1) + 2m(H)) - (t(H) + 2m(H)) \\ &= O(D(n)) + O(t(H)) - t(H) \\ &= O(D(n)). \end{aligned}$$

لأنَّ بإمكاننا رفع قيمة وحدات الكومون لطغنى على الثابت المضمّن في  $O(t(H))$ . حدسيًا، تسدد تكلفة إجراء كل رابط من تقليص الكومون نظرًا لأنَّ الرابط ينقص عدد الجذور بمقدار واحد. سنرى في المقطع 4.19 أنَّ  $D(n) = O(\lg n)$ ، فتكون التكلفة للمخمّدة لاستخراج العقدة الصغرى  $O(\lg n)$ .

## تمارين

## 1-2.19

اعرض كومة فيبوناتشي الناتجة عن استدعاء FIB-HEAP-EXTRACT-MIN على كومة فيبوناتشي المعروضة في الشكل 4.19(ش).

## 3.19 إنقاص قيمة مفتاح وحذف عقدة

في هذا المقطع، نبين كيفية إنقاص قيمة مفتاح عقدة في كومة فيبوناتشي بزمن محدد  $O(1)$  وكيفية حذف أية عقدة من كومة فيبوناتشي ذات  $n$  عقدة بزمن محدد  $O(D(n))$ . سنثبت في المقطع 4.19 أنَّ الدرجة العظمى  $D(n)$  هي  $O(\lg n)$ ، وهذا يعني أن تنفيذ كلٍّ من FIB-HEAP-EXTRACT-MIN و FIB-HEAP-DELETE يتم بزمن محدد  $O(\lg n)$ .

## إنقاص قيمة مفتاح

في شبه الرمز التالي للعملية FIB-HEAP-DECREASE-KEY، نفترض - كما أسلفنا - أنَّ حذف عقدة من لائحة مترابطة لا يغير أي من الواصفات البنيوية في العقدة المحذوفة.

FIB-HEAP-DECREASE-KEY( $H, x, k$ )

```

1  ■  $k > x.key$ 
2  error "new key is greater than current key"
3   $x.key = k$ 
4   $y = x.p$ 
5  ■  $y \neq \text{NIL}$  and  $x.key < y.key$ 
6    CUT( $H, x, y$ )
7    CASCADING-CUT( $H, y$ )
8  ■ if  $x.key < H.min.key$ 
9     $H.min = x$ 
```

CUT( $H, x, y$ )

```

1  remove  $x$  from the child list of  $y$ , decrementing  $y.degree$ 
2  add  $x$  to the root list of  $H$ 
3   $x.p = \text{NIL}$ 
4   $x.mark = \text{FALSE}$ 
```

CASCADING-CUT( $H, y$ )

```

1   $z = y.p$ 
2  if  $z \neq \text{NIL}$ 
3    if  $y.mark == \text{FALSE}$ 
```



```

4      y.mark = TRUE
5      else CUT(H, y, z)
6      CASCADING-CUT(H, z)

```

يعمل إجراء **FIB-HEAP-DECREASE-KEY** كما يلي. تضمن الأسطر 1-3 ألا يكون المفتاح الجديد أكبر من المفتاح الحالي للعقدة  $x$ ، ثم تسند للمفتاح الجديد إلى  $x$ . إذا كان  $x$  جذرًا أو كان  $x.key \geq y.key$  حيث  $y$  هو أبو  $x$ ، فلا حاجة إلى تغييرات بنوية لأن ترتيب الكومة وفق الأصغر لم يُخرق. تختبر الأسطر 4-5 هذا الشرط.

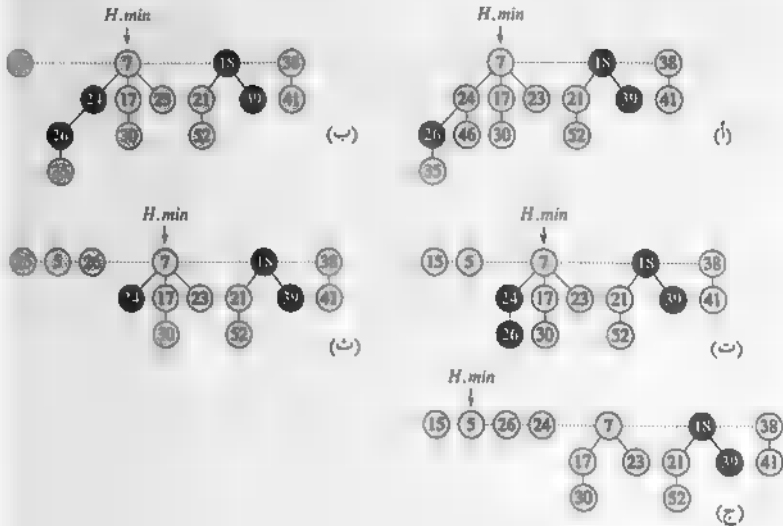
إذا جرى خرق ترتيب الكومة وفق الأصغر، فقد تحدث عدة تغييرات. نبدأ بقطع  $x$  في السطر 6. "يقطع" الإجراء **CUT** الرابط بين  $x$  وأبها  $y$  جاعلاً، من  $x$  جذرًا. نستخدم الواصفات **mark** للوصول إلى الحدود الزمنية المرغوبة، فهي تُسجّل جزءًا صغيرًا من تاريخ كل عقدة. افترض أن الأحداث التالية وقعت للعقدة  $x$ :

1. في وقت ما كانت  $x$  جذرًا،
2. ثم رُبطت  $x$  بعقدة أخرى (مكوّنةً ابنًا لها)،
3. ثم جرى حذف ابنُ  $x$  للعقدة  $x$  بالقطع.

بمجرد فقدان الابن الثاني، انفصل  $x$  عن أبيها جاعلين منها جذرًا جديدًا. تكون قيمة الواصفة  $x.mark$  مساوية **TRUE** إذا حدثت الخطوتان 1 و 2 وجرى قطع ابن واحد للعقدة  $x$ . فيقوم الإجراء **CUT** بمسح  $x.mark$  في السطر 4 لأنه يُنفَّذ الخطوة 1. (يمكننا الآن معرفة سبب مسح **clear** السطر 3 من **FIB-HEAP-LINK** للقيمة  $y.mark$ : فالعقدة  $y$  جرى ربطها بعقدة أخرى وبذلك تُنفَّذ الخطوة 2. في المرة التالية التي يجري فيها قطع أحد أبناء  $y$  ستصبح قيمة  $y.mark$  هي **TRUE**.)

لم تنته بعد، لأن  $x$  قد تكون الابن الثاني للفصل عن الأب  $y$  منذ ربط  $y$  بعقدة أخرى. لذلك، يحاول السطر 7 من **FIB-HEAP-DECREASE-KEY** إجراء عملية **قطع متتابع cascading-cut** على  $y$ . إذا كانت  $y$  جذرًا فسيُسبّب الاختبار في السطر 2 من **CASCADING-CUT** عودة الإجراء فقط. إذا لم تكن  $y$  معلّمة يقوم الإجراء بتعليمها في السطر 4، لأن ابنها الأول قد قُطع للتو، ثم يعود. لكن إذا كانت  $y$  معلّمة، فهي قد فقدت للتو ابنها الثاني؛ فيجري قطع  $y$  في السطر 3 ويستدعى الإجراء **CASCADING-CUT** نفسه عوديًا في السطر 6 على  $y$  أبي  $y$ . يستمر الإجراء **CASCADING-CUT** بالتنفيذ العودي إلى أعلى الشجرة حتى يصل إلى جذر أو إلى عقدة غير معلّمة.

عند انتهاء جميع عمليات القطع المتتامة، يُنهي السطران 8-9 من **FIB-HEAP-DECREASE-KEY** العمل بتحديث  $H.min$  إذا كان ذلك ضروريًا. العقدة الوحيدة التي جرى تغيير مفتاحها كانت هي العقدة  $x$



**الشكل 5.19** استدعاء ان للإجراء FIB-HEAP-DECREASE-KEY. (أ) كومة فيونانشي في البداية. (ب) جرى إنقاص مفتاح العقدة ذات للمفتاح 46 إلى 15. تصبح هذه العقدة جذراً، ويجري تعليم أبيها (العقدة ذات للمفتاح 24) الذي لم يكن معلماً. (ت) - (ج) يجري إنقاص مفتاح العقدة ذات للمفتاح 35 إلى 5. في الجزء (ت) تصبح العقدة ذات للمفتاح 5 جذراً. يجري تعليم أبيها ذي للمفتاح 26، فيحدث قطع متتابع. يجري فصل العقدة 26 عن أبيها وجعلها جذراً غير معلّم في (ت). يحدث قطع متتابع آخر لأن العقدة ذات للمفتاح 24 هي أيضاً معلّمة. فيجري فصلها عن أبيها وجعلها جذراً غير معلّم في الجزء (ج). يتوقف القطع للتتابع عند هذه النقطة لأن العقدة ذات للمفتاح 7 هي جذر. (حتى إن لم تكن هذه العقدة جذراً فيستوقف القطع للتتابع لأنها غير معلّمة). تظهر نتيجة عملية FIB-HEAP-DECREASE-KEY في الجزء (ج)، مع المؤشر  $H.min$  الذي يشير إلى العقدة الصغرى الجديدة.

التي جرى إنقاص قيمة مفتاحها. لذلك، فالعقدة الصغرى الجديدة هي إما العقدة الصغرى الأصلية وإما العقدة  $x$ .

يُظهر الشكل 5.19 تنفيذ استدعاءين للإجراء FIB-HEAP-DECREASE-KEY بدءاً من كومة فيونانشي الظاهرة في الشكل 5.19 (أ). لا يتطلب الاستدعاء الأول المبيّن في الشكل 5.19 (ب) قطعاً متتابعاً. أما الاستدعاء الثاني المبيّن في الشكل 5.19 (ت) - (ج) فهو يستدعي عمليتي قطع متابعتين.

سنبيّن الآن أن التكلفة للمعمّدة للإجراء FIB-HEAP-DECREASE-KEY هي  $O(1)$  فقط. نبدأ بتحديد التكلفة الفعلية. يأخذ الإجراء FIB-HEAP-DECREASE-KEY زمناً  $O(1)$ ، إضافةً إلى زمن إجراء القطع للتتابع. افترض أنّ الإجراء FIB-HEAP-DECREASE-KEY استدعى الإجراء CASCADING-CUT عودياً  $c$

مرة في طلب ما (الطلب المنشأ في السطر 7 من FIB-HEAP-DECREASE-KEY متبوعًا بـ  $c-1$  طلب عودي للإجراء CASCADING-CUT). يتطلب كل استدعاء للإجراء CASCADING-CUT زمنًا  $O(1)$  بدون الطلبات العودية. فتكون التكلفة الفعلية للإجراء FIB-HEAP-DECREASE-KEY مع جميع الطلبات العودية هي  $O(c)$ .

نحسب فيما يلي التغير في الكمون. لتكن  $H$  كومة فيوناتشي قبل عملية FIB-HEAP-DECREASE-KEY مباشرة. إن طلب CUT في السطر 6 من FIB-HEAP-DECREASE-KEY يُنشئ شجرة جديدة جذرها العقدة  $x$  ويمسح خانة العلام في  $x$  (التي يمكن أن تكون FALSE سلفًا). يقطع كل استدعاء عودي لـ CASCADING-CUT، عدا الطلب الأخير، عقدة معلّمة ويمسح خانة التعليم. فتحتوي كومة فيوناتشي بعد التنفيذ  $c + t(H)$  شجرة (الأشجار الأصلية  $t(H)$  و  $c - 1$  شجرة ناتجة عن القطع المتتابع، والشجرة التي جذرها  $x$ ) و  $m(H) - c + 2$  عقدة معلّمة على الأكثر  $c - 1$  عقدة أزله تعليمها في القطع المتتابع وقد يكون الاستدعاء الأخير لـ CASCADING-CUT قد علّم عقدة. فيكون بذلك التغير في الكمون هو على الأكثر

$$((t(H) + c) + 2(m(H) - c + 2)) - (t(H) + 2m(H)) = 4 - c.$$

فتكون التكلفة المحققة لـ FIB-HEAP-DECREASE-KEY هي على الأكثر

$$O(c) + 4 - c = O(1).$$

لأن بإمكاننا رفع قيمة وحدات الكمون لتغطي على الثابت للضئ في  $O(c)$ . يمكن أن نثرف الآن سبب تعريف دالة الكمون ليتضمن حدًا هو ضعف عدد العقد المعلّمة. عندما يجري قطع عقدة معلّمة  $y$  في قطع متابع، يُمنح تعليمها، فينقص الكمون بمقدار 2. تُدفع وحدة من الكمون للقطع وتُمنح بت التعليم، وتُصرف الوحدة الأخرى لزيادة وحدة الكمون بسبب تحول  $y$  إلى جذر.

### حذف عقدة

يحذف شبه الرماز التالي عقدة من كومة فيوناتشي ذات  $n$  عقدة بزمن محدد  $O(D(n))$ . نفترض أنه لا يوجد حاليًا مفتاح قيمته  $-\infty$  في كومة فيوناتشي.

FIB-HEAP-DELETE( $H, x$ )

- 1 FIB-HEAP-DECREASE-KEY ( $H, x, -\infty$ )
- 2 FIB-HEAP-EXTRACT-MIN( $H$ )

يُصير الإجراء FIB-HEAP-DELETE العقدة  $x$  العقدة الصغرى في كومة فيوناتشي عن طريق إعطائها مفتاحًا فريدًا في صغره  $-\infty$ . ثم يُحذف الإجراء FIB-HEAP-EXTRACT-MIN العقدة  $x$  من كومة فيوناتشي. إن

الزمن للمخحد ل FIB-HEAP-DELETE هو مجموع الزمن للمخحد ل FIB-HEAP-DECREASE-KEY وهو  $O(1)$  والزمن للمخحد ل FIB-HEAP-EXTRACT-MIN وهو  $O(D(n))$  وسنرى في لقطع 4.19 أنَّ  $D(n) = O(\lg n)$ ، لذا فإن الزمن للمخحد ل FIB-HEAP-DELETE هو  $O(\lg n)$ .

#### تمارين

##### 1-3.19

افترض أنَّ جذرًا  $x$  في كومة فيوناتشي معلّم. اشرح كيف أصبح  $x$  جذرًا معلّمًا. بيّن أنه ليس من المهم التحليل كون  $x$  معلّمًا، حتى لو لم يكن  $x$  جذرًا قد رُبط أولاً بعقدة أخرى ثم فُقد ابنًا واحدًا.

##### 2-3.19

برّر الزمن للمخحد  $O(1)$  لإجراء FIB-HEAP-DECREASE-KEY باعتباره تكلفة وسطى للعمليات باستخدام تحليل مجّمع aggregate analysis.

## 4.19 وضع حد للدرجة العظمى

لإثبات أنَّ الزمن للمخحد ل FIB-HEAP-DELETE و FIB-HEAP-EXTRACT-MIN هو  $O(\lg n)$ ، يجب أن نُظهر أنَّ الحد الأعلى  $D(n)$  للدرجة أية عقدة في كومة فيوناتشي ذات  $n$  عقدة هو  $O(\lg n)$ . سنبين خصوصًا أنَّ  $D(n) \leq \lceil \log_\phi n \rceil$  حيث  $\phi$  هي النسبة الذهبية للعزّة في المعادلة (24.3) كما يلي

$$\phi = (1 + \sqrt{5})/2 = 1.61803 \dots$$

يكون مبدأ التحليل كالتالي. لكل عقدة  $x$  في كومة فيوناتشي، نُعرّف  $\text{size}(x)$  على أنه عدد العقد في الشجرة الفرعية ذات الجذر  $x$  ومنها العقدة  $x$  نفسها. (لاحظ أنه ليس من الضروري أن تكون  $x$  ضمن لائحة الجذور بل قد تكون أية عقدة.) سنبين أنَّ  $\text{size}(x)$  يزداد أسّيًا بحسب  $x.\text{degree}$ . استحضّر في ذهنك أنه تجري المحافظة على  $x.\text{degree}$  دومًا بحيث تساوي العدد الدقيق للدرجة  $x$ .

#### مبرهنة 1.19

لتكن  $x$  عقدة في كومة فيوناتشي، ولنفترض أنَّ  $x.\text{degree} = k$ . ولتكن  $y_1, y_2, \dots, y_k$  أبناء  $x$  بترتيب ربطها بـ  $x$  من الأقدم إلى الأحدث. فيكون عند ذلك  $y_1.\text{degree} \geq 0$  و  $y_i.\text{degree} \geq i - 2$  لكل  $i = 2, 3, \dots, k$ .

**البرهان** من البديهي أنَّ  $y_1.\text{degree} \geq 0$ .

في حالة  $i \geq 2$ ، نلاحظ أنه عندما كانت  $y_i$  مرتبطة مع  $x$ ، فإن جميع  $y_1, y_2, \dots, y_{i-1}$  كانت أبناء

لـ  $x$ ، ولا بد أنه كان لدينا  $x.degree \geq i-1$ . ولما كانت العقدة  $y_i$  ترتبط مع العقدة  $x$  (باستخدام CONSOLIDATE) فقط إذا كانت  $x.degree = y_i.degree$ ، فيجب أن يكون لدينا أيضًا  $y_i.degree \geq i-1$  في ذلك الوقت. ومنذ ذلك الوقت فقدت  $y_i$  على الأكثر ابنًا واحدًا، لأنها لو كانت فقدت ابنين لجرى قطعها من  $x$  (باستخدام CASCADING-CUT). وبذلك نستنتج أن

$$y_i.degree \geq i-2$$

وصلنا أخيرًا إلى الجزء التحليلي الذي يشرح الاسم "كومات فيوناتشي". تذكر من المقطع 2.3 أنه في حالة  $k = 0, 1, 2, \dots$  يُعرّف عدد فيوناتشي من المرتبة  $k$  عوديًا بالشكل:

$$F_k = \begin{cases} 0 & \text{if } k = 0, \\ 1 & \text{if } k = 1, \\ F_{k-1} + F_{k-2} & \text{if } k \geq 2. \end{cases}$$

تقدّم التوطئة الآتية طريقةً أخرى للتعبير عن  $F_k$ .

### توطئة 2.19

$$F_{k+2} = 1 + \sum_{i=0}^k F_i.$$

لكل الأعداد الصحيحة  $k \geq 0$

**البرهان** يجري البرهان بالتدريج على  $k$ . فعندما تكون  $k = 0$

$$\begin{aligned} 1 + \sum_{i=0}^0 F_i &= 1 + F_0 \\ &= 1 + 0 \\ &= F_2. \end{aligned}$$

نفترض الآن الفرض التدرجي  $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$ ، ويكون لدينا

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &= F_k + \left( 1 + \sum_{i=0}^{k-1} F_i \right) \\ &= 1 + \sum_{i=0}^k F_i. \end{aligned}$$

■

### توطئة 3.19

يحقق عدد فيوناتشي ذو الترتيب  $(k+2)$  المعادلة  $F_{k+2} \geq \phi^k$ ، لكل الأعداد الصحيحة  $k \geq 0$ .

**البرهان** يجري البرهان بالتدريج على  $k$ . الحالتان الأساسيتان هما  $k = 0$  و  $k = 1$ . عندما تكون  $k = 0$  يكون لدينا  $F_2 = 1 = \phi^0$ ، وعندما يكون  $k = 1$  يكون لدينا  $F_3 = 2 > 1.619 > \phi^1$ . الخطوة التدرجية لكل  $k \geq 2$ ، ونفترض أن  $F_{i+2} > \phi^i$  لكل  $i = 0, 1, \dots, k-1$ . نذكر أن  $\phi$  هو الجذر الموجب للمعادلة (23.3)،  $x^2 = x + 1$ . فيكون لدينا

$$\begin{aligned} F_{k+2} &= F_{k+1} + F_k \\ &\geq \phi^{k-1} + \phi^{k-2} \quad (\text{بحسب فرضية التدرج}) \\ &= \phi^{k-2}(\phi + 1) \\ &= \phi^{k-2} \cdot \phi^2 \quad ((\text{بحسب المعادلة (23.3)}) \\ &= \phi^k. \end{aligned}$$

■

يُكتمل التحليل بالتوسطة التالية مع نتيجتها.

#### توسطة 4.19

تكن  $x$  عقدة ما في كومة فيبوناتشي، وليكن  $k = x.degree$ ، فيكون  $\text{size}(x) \geq F_{k+2} \geq \phi^k$  حيث  $\phi = (1 + \sqrt{5})/2$ .

**البرهان** نرمز بـ  $s_k$  للحجم الأصغر الممكن لأية عقدة من الدرجة  $k$  في أية كومة فيبوناتشي. من البديهي أن  $s_0 = 1$  و  $s_1 = 2$ . العدد  $s_k$  هو على الأكثر  $\text{size}(x)$ ، ولأن إضافة أبناء إلى عقدة ما لا يمكن أن يقلل من حجمها، فإن قيمة  $s_k$  تزداد بانتظام بزيادة  $k$ . افترض أن عقدة ما ■ في كومة فيبوناتشي تحقق  $\text{size}(z) = s_k$  و  $z.degree = k$ . ولما كان  $s_k \leq \text{size}(x)$  فإننا نحسب حدًا أدنى على  $\text{size}(x)$  بحساب حد أدنى على  $s_k$ . وكما في التوسطة 1.19، لتكن  $y_1, y_2, \dots, y_k$  هي أبناء  $z$  بترتيب ربطها بـ  $z$ . لحساب حد أصغر للقيمة  $s_k$  نحسب واحدًا من أجل  $z$  نفسها وواحدًا من أجل الابن الأول  $y_1$  (والذي يحقق  $\text{size}(y_1) \geq 1$ ) فيكون

$$\begin{aligned} \text{size}(x) &\geq s_k \\ &\geq 2 + \sum_{i=2}^k s_{y_i.degree} \\ &\geq 2 + \sum_{i=2}^k s_{i-2}, \end{aligned}$$

حيث ينتج السطر الأخير من تطبيق التوسطة 1.19 (وبذلك يكون  $i - 2$  و  $y_i.degree$ ) ومن التزايد المنتظم  $s_k$  (وبذلك يكون  $s_{i-2} \geq s_{y_i.degree}$ ).

نبين الآن بالتدريج على  $k$  أنَّ  $s_k \geq F_{k+2}$  لكل الأعداد الصحيحة الموجبة  $k$ . الحالتان الأساسيتان  $k=0$  و  $k=1$  بديهيتان. نفترض من أجل الخطوة التدرجية أنَّ  $k \geq 2$  وأنَّ  $s_l \geq F_{l+2}$  لكل  $l = 0, 1, \dots, k-1$ ، فيكون لدينا:

$$\begin{aligned}
 s_k &\geq 2 + \sum_{l=2}^k s_{l-2} \\
 &\geq 2 + \sum_{l=2}^k F_l \\
 &= 1 + \sum_{l=0}^k F_l \\
 &= F_{k+2} \quad (\text{بحسب التوطئة (2.19)}) \\
 &\geq \phi^k \quad (\text{بحسب التوطئة (3.19)})
 \end{aligned}$$

■ وبذلك نكون قد أثبتنا أنَّ  $\text{size}(x) \geq s_k \geq F_{k+2} \geq \phi^k$ .

### نتيجة 5.19

الدرجة العظمى  $D(n)$  لعقدة ما في كومة فيوناتشي ذات  $n$  عقدة هي  $O(\lg n)$ .

**البرهان** لكن  $x$  عقدة ما في كومة فيوناتشي ذات  $n$  عقدة، ولتكن  $k = x.\text{degree}$ . بحسب التوطئة 4.19 يكون  $n \geq \text{size}(x) \geq \phi^k$ . بأخذ اللوغاريتم ذي الأساس  $\phi$  يكون  $k \leq \log_{\phi} n$ . (في الحقيقة،  $\lfloor \log_{\phi} n \rfloor \leq k$  لأنَّ  $k$  عدد صحيح.) فتكون بذلك الدرجة العظمى  $D(n)$  لأي عقدة هي  $O(\lg n)$ . ■

### تمارين

#### 1-4.19

يؤكد الأستاذ Pinocchio أنَّ ارتفاع كومة فيوناتشي ذات  $n$  عقدة هو  $O(\lg n)$ . بَيِّنْ أن الأستاذ مخطئ من خلال عرض متتالية من عمليات كومات فيوناتشي تُنشئ كومة فيوناتشي مؤلفة من شجرة واحدة فقط على شكل سلسلة خطية من  $n$  عقدة، لأي عدد  $n$  صحيح موجب.

#### 2-4.19

افترض أننا نعلم قاعدة القطع للتابع بحيث نقطع عقدة  $x$  من أيها مباشرةً بعد فقدانها الابن ذا الترتيب  $k$ ، حيث  $k$  عدد ثابت صحيح. (تستخدم القاعدة في اللقطع 3.19 القيمة  $k=2$ .) ما هي قيم  $k$  التي تحقق  $D(n) = O(\lg n)$ ؟

## مسائل

## I-19 تنجيز بديل للمحذف

اقترح الأستاذ PISANO الشكل التالي للإجراء FIB-HEAP-DELETE، مدعيًا أنه يعمل بسرعة أكبر عندما لا تكون العقدة المحذوفة هي العقدة التي يشير إليها  $H.min$ .

PISANO-DELETE( $H, x$ )

```

1  if  $x == H.min$ 
2    FIB-HEAP-EXTRACT-MIN( $H$ )
3  else  $y = x.p$ 
4    if  $y \neq NIL$ 
5      CUT( $H, x, y$ )
6      CASCADING-CUT( $H, y$ )
7    add  $x$ 's child list to the root list of  $H$ 
8    remove  $x$  from the root list of  $H$ 
```

أ. إن ادعاء الأستاذ بأن هذا الإجراء يُنفذ بسرعة أكبر يعتمد جزئيًا على افتراض أن السطر 7 يمكن أن يُنفذ بزمن فعلي  $O(1)$ . ما هو الخطأ في هذا الافتراض؟

ب. أعط حلاً أعلى جيدًا للزمن الفعلي للإجراء PISANO-DELETE عندما لا تكون  $x$  هي  $H.min$ . يجب أن يكون الحد الأعلى بدلالة  $x.degree$  والعدد  $c$  الذي يمثل عدد استدعاءات الإجراء CASCADING-CUT.

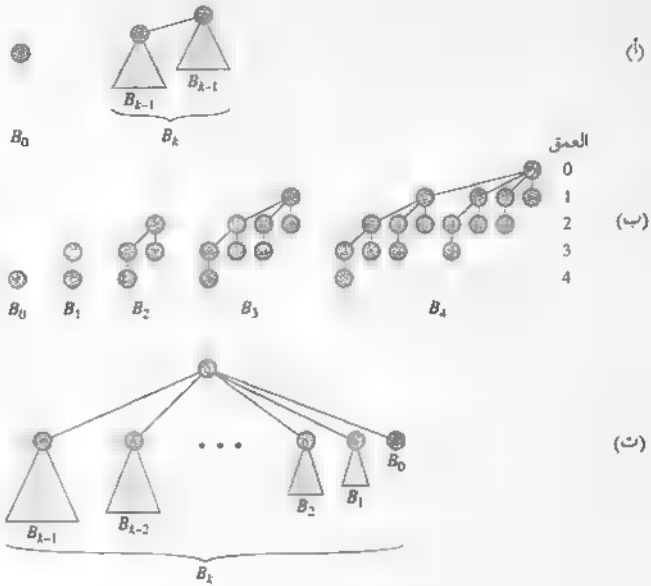
ج. افترض أننا نستخدم الإجراء PISANO-DELETE( $H, x$ )، ولكن  $H'$  كومة فيوناتشي الناتجة. بافتراض أن العقدة  $x$  ليست جذرًا، ضع حدًا لكمون  $H'$  بدلالة  $x.degree$  و  $c$  و  $t(H)$  و  $m(H)$ .

د. استنتج أن الزمن للمحذف للإجراء PISANO-DELETE ليس أفضل بالمقارنة من زمن FIB-HEAP-DELETE، حتى عندما تكون  $x \neq H.min$ .

## 2-19 أشجار ثنائية الحد وكومات ثنائية الحد

الشجرة الثنائية الحد  $B_k$  هي شجرة مرتبة (انظر للقطع ب.2.5) تُعرف تعريفًا عوديًا. وكما يظهر في الشكل 6.19(أ)، تتألف الشجرة الثنائية الحد  $B_0$  من عقدة وحيدة. وتتألف الشجرة الثنائية الحد  $B_k$  من شجرتين ثنائيتين الحد  $B_{k-1}$  مرتبطتان إحداها بالأخرى: جذر إحداها هو الابن في أقصى اليسار لجذر الأخرى. يبين الشكل 6.19(ب) الأشجار الثنائية الحد من  $B_0$  إلى  $B_4$ .





الشكل 6.19 (أ) التعريف العودي للشجرة الثنائية الحد  $B_k$ . تُعَمَّل للثلاث أشجارًا فرعية ذات جذور. (ب) الأشجار الثنائية الحد من  $B_0$  إلى  $B_4$ ، تُظهر أعماق العقد في  $B_4$ . (ت) طريقة أخرى للنظر إلى الشجرة الثنائية الحد  $B_k$ .

أ. أثبت أنه في شجرة ثنائية الحد  $B_k$

1. توجد  $2^k$  عقدة.

2. ارتفاع الشجرة هو  $k$ .

3. هناك تمامًا  $\binom{k}{i}$  عقدة في العمق  $i$  لكل  $i = 0, 1, \dots, k$ .

4. درجة الجذر هي  $k$ ، وهي أكبر من درجة أية عقدة أخرى؛ إضافة إلى أنه، كما يُظهر

الشكل 6.19(ت)، إذا كانت أرقام أبناء الجذر من اليسار إلى اليمين:  $0, k-2, \dots, k-1$ ، فإن

الابن  $i$  هو جذر للشجرة الفرعية  $B_i$ .

**الكومة الثنائية الحد**  $\text{binomial heap } H$  هي مجموعة من الأشجار الثنائية الحد تحقق الخصائص الآتية:

1. كل عقدة لها مفتاح (كما في كومات فيوناتشي).

2. كل شجرة ثنائية الحد في  $H$  تخضع لخصائص الكومات للرتبة وفق الأصغر  $\text{min-heap property}$ .

3. في حالة أي عدد  $k$  صحيح غير سالب، يوجد على الأكثر شجرة ثنائية الحد في  $H$  جذرها من الدرجة  $k$ .

ب. افترض أن كومة ثنائية الحد  $H$  فيها  $n$  عقدة. ناقش العلاقة بين الأشجار الثنائية الحد التي تحتويها  $H$  والتعميل الاثنائي لـ  $n$ . استنتج أن  $\lceil \lg n \rceil + 1$  تتألف من شجرة ثنائية الحد.

افترض أننا نغزل كومة ثنائية الحد كالتالي. يُغزل أسلوب اليمين الأيسر والأخ الأيمن للمقدم في المقطع 4.10 كل شجرة ثنائية الحد ضمن كومة ثنائية الحد. تحتوي كل عقدة مفتاحها ومؤشر إلى أبنائها، وآخر على ابنها في أقصى اليسار، وثالث إلى أخيهما الأيمن المباشر (هذه المؤشرات تكون NIL عند اللزوم)؛ ودرجتها (كما في كومات فيوناتشي) عدد أولادها. تكون الجذور لائحة جذور متزايدة، مرتبة بحسب درجات الجذور (من الأدنى إلى الأعلى)، ويجري النفاذ إلى الكومة الثنائية الحد عن طريق مؤشر إلى العقدة الأولى في لائحة الجذور.

ت. أكمل وصف كيفية تمثيل كومة ثنائية الحد (سَمِّ الوصفات، وحدد متى تأخذ الوصفات القيمة NIL، وعرف كيفية تنظيم لائحة الجذور)، وأظهر كيفية تنجيز العمليات السبع نفسها على الكومات الثنائية الحد كما يُجرى هذا الفصل على كومات فيوناتشي. يجب أن تُنفذ كل عملية بزمن  $O(\lg n)$  في أسوأ الحالات، حيث  $n$  هو عدد العقد في الكومة الثنائية الحد (أو في حالة عملية UNION، في الكومتين الثنائيتين اللتين يجري جمعهما). يجب أن تستغرق عملية MAKE-HEAP زمناً ثابتاً.

ث. افترض أننا نريد تنجيز العمليات على الكومات القابلة للدمج فقط في كومات فيوناتشي (أي لا نُشعر عمليتي DECREASE-KEY أو DELETE). كيف يمكن أن تشبه الأشجار في كومة فيوناتشي تلك الموجودة في كومة ثنائية الحد؟ بماذا تختلف عنها؟ بين أن الدرجة العظمى في كومة فيوناتشي ذات  $n$  عقدة ستكون  $\lceil \lg n \rceil$  على الأكثر.

ج. اخترع للمدرس McGee بنية معطيات جديدة تعتمد على كومات فيوناتشي. كومة McGee لها بنية كومة فيوناتشي نفسها وتدعم عمليات الكومات القابلة للدمج فقط. وطريقة تنجيز العمليات هي نفسها في كومات فيوناتشي، إلا أن الإدراج والاجتماع يُدعمان لائحة الجذور في خطواتها الأخيرة. ما هي أزمّة تنفيذ العمليات على كومات McGee في أسوأ الحالات ؟

### 3-19 المزيد من العمليات على كومات فيوناتشي

نرغب بإغناء كومة فيوناتشي  $H$  لتدعم عمليتين جديدتين دون تغيير زمن التنفيذ للمحدد لأية عملية أخرى على كومات فيوناتشي.

أ. العملية  $\text{FIB-HEAP-CHANGE-KEY}(H, x, k)$  تُغيّر مفتاح العقدة  $x$  إلى القيمة  $k$ . أعطِ تنجيروًا فعالاً لهذا الإجراء، وحلّل زمن التنفيذ للمخحد لهذا التنجيز في الحالات التي تكون فيها  $k$  أكبر من المفتاح  $x.key$ ، أو أصغر منه، أو مساوية له.

ب. أعطِ تنجيروًا فعالاً للإجراء  $\text{FIB-HEAP-PRUNE}(H, r)$  الذي يحذف  $q = \min(r, H.n)$  عقدة من  $H$ . يمكن أن تختار أي  $q$  عقدة لتحذفها. حلّل زمن التنفيذ للمخحد لتنجيزك. (تلميح: قد تحتاج إلى تعديل بنية المعطيات ودالة الكمون.)

#### 4-19 الكومات 2-3-4

تدّم الفصل 18 الشجرة 2-3-4 التي يكون لكل عقدة داخلية فيها (ربما ماعدا الجذر) ابنان أو ثلاثة أو أربعة أبناء ولكل الأوراق العمق نفسه. في هذه المسألة سنقوم بتنجيز الكومات 2-3-4، التي تدعم العمليات على الكومات القابلة للدمج.

تختلف الكومات 2-3-4 عن الأشجار 2-3-4 في المناحي التالية: في الكومات 2-3-4، الأوراق فقط هي التي تخزّن المفاتيح، وكل ورقة  $x$  تخزن مفتاحاً واحداً فقط في الوصفة  $x.key$ . يمكن أن تظهر المفاتيح في الأوراق بأي ترتيب. تحتوي كل عقدة داخلية  $x$  قيمة  $x.small$  تساوي أصغر مفتاح مخزن في أي ورقة في شجرة فرعية جذرها  $x$ . يحتوي الجذر  $r$  واصفة  $r.height$  هو ارتفاع الشجرة. أخيراً، الكومات 2-3-4 مخدّدة لإبقائها في الذاكرة الرئيسية بحيث لا نحتاج إلى القراءة من القرص أو الكتابة عليه.

يُجرّ العمليات التالية على الكومات 2-3-4. في الأجزاء (أ)–(ج)، أي عملية يجب أن تنفذ بزمن  $O(\lg n)$  على كومات 2-3-4 ذات  $n$  عنصرًا. عملية UNION في الجزء (و) يجب أن تنفذ بزمن  $O(\lg n)$ ، حيث  $n$  هو عدد العناصر في كومتَي الدخل.

أ. العملية MINIMUM التي تعيد مؤشرًا إلى الورقة التي تحتوي للمفتاح الأصغري.

ب. العملية DECREASE-KEY التي تنقص مفتاح ورقة معينة  $x$  إلى قيمة محدّدة  $k \leq x.key$ .

ت. العملية INSERT التي تدرج ورقة  $x$  مفتاحها  $k$ .

ث. العملية DELETE التي تحذف ورقة معينة  $x$ .

ج. العملية EXTRACT-MIN التي تزعج الورقة ذات للمفتاح الأصغري.

ح. العملية UNION التي توحد كومتين من نوع 2-3-4، وتعيد كومة واحدة 2-3-4، وتدمّر كومتَي الدخل.

## ملاحظات الفصل

أدخل Fredman و Tarjan [114] كومات فيوناتشي. نصف هذه المقالة أيضًا تطبيق كومات فيوناتشي على مسائل أقصر المسارات من منبع وحيد، وأقصر المسارات من أية عقدة إلى أية عقدة، وللزوجة الثنائية الجزء المثقلة، ومسألة شجرة المسح الصغرى.

بعد ذلك، طوّر Driscoll و Gabow و Shraiman و Tarjan [97] "الكومات المرخاة" relaxed heaps كبديل عن كومات فيوناتشي. وحدّدوا صنفان من الكومات المرخاة. يعطي أحدهما حدود كومات فيوناتشي الزمنية للمعمّدة نفسها. ويسمح الآخر بتنفيذ DECREASE-KEY بزمن  $O(1)$  في أسوأ الحالات (غير معمّدة)، وتنفيذ EXTRACT-MIN و DELETE بزمن  $O(\lg n)$  في أسوأ الحالات. كذلك فإنّ للكومات المرخاة بعض الميزات على كومات فيوناتشي في الخوارزميات المتوازنة.

ارجع أيضًا إلى ملاحظات الفصل 6 فقيها معلومات عن بنى معطيات أخرى تدعم عمليات DECREASE-KEY سريعة عندما تكون متتالية القيم التي تعيدها استدعاءات EXTRACT-MIN متزايدة بانتظام عبر الزمن، وتكون المعطيات أعدادًا صحيحة في مجال محدد.

شاهدنا في فصول سابقة بنى معطيات تدعم عمليات الأرتال ذات الأولوية: الكومات الثنائية في الفصل 6، والأشجار الحمراء-السوداء في الفصل 13،<sup>1</sup> وكومات فيبوناتشي في الفصل 19. ورأينا أنَّ في كلِّ بنى من بنى المعطيات هذه، توجد عملية هامة واحدة على الأقل تستغرق زمنًا  $O(\lg n)$ ، إما في أسوأ الحالات وإما في الحالة المخمدة. والواقع أنه لما كانت جميع بنى المعطيات هذه تعتمد في قراراتها على مقارنة المفاتيح، فإن الحد الأدنى للفرز  $\Omega(n \lg n)$ ، الوارد في اللقطع 1.8، يعني أنَّ عملية واحدة على الأقل يجب أن تستغرق زمنًا  $\Omega(\lg n)$ . لماذا؟ إذا استطعنا إجراء عمليتي INSERT و EXTRACT-MIN بزمن  $O(\lg n)$ ، يمكننا عندها فرز  $n$  مفتاحًا بزمن  $O(n \lg n)$  بإجراء  $n$  عملية INSERT أولاً، ثم  $n$  عملية EXTRACT-MIN.

غير أننا شاهدنا في الفصل 8، أن بإمكاننا أحيانًا استغلال معلومات إضافية عن المفاتيح لإجراء الفرز بزمن  $O(n \lg n)$ . ويمكننا، بصورة خاصة في الفرز بالعد، فرز  $m$  مفتاحًا، كلٌّ منها هو عدد صحيح يقع ضمن المجال من 0 إلى  $k$ ، بزمن  $\Theta(n + k)$ ، والذي هو  $\Theta(n)$  في حال كانت  $k = O(n)$ .

ولما كان باستطاعتنا الالتفاف حول الحد الأدنى للفرز  $\Omega(n \lg n)$  عندما نكون نكوّن المفاتيح أعدادًا صحيحة ضمن مجال محدود، يمكنك أن تتساءل: هل نستطيع، بأسلوب مشابه، إجراء كلِّ من عمليات الزل ذي الأولوية بزمن  $O(\lg n)$ ؟ سنرى في هذا الفصل أن ذلك ممكن: إذ إنَّ أشجار Van Emde Boas تدعم عمليات الزل ذي الأولوية، وبعض العمليات الأخرى بزمن  $O(\lg \lg n)$  في أسوأ الحالات. الفكرة هنا هي أن المفاتيح يجب أن تكون أعدادًا صحيحة ضمن المجال الممتد من 0 إلى  $n - 1$ ، دون السماح بتكرار أيٍّ منها.

تدعم أشجار Van Emde Boas، على وجه الخصوص، كلاً من العمليات الآتية على المجموعات الديناميكية المسرودة في الصفحة 230 وهي: SEARCH و INSERT و DELETE و MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR، وذلك بزمن  $O(\lg \lg n)$ . ولن نناقش، في هذا الفصل، للمعطيات التابعة،

<sup>1</sup> لا يناقش الفصل 13 صراحة كيفية تحجيز EXTRACT-MIN و DECREASE-KEY، ولكن بإمكاننا بناء هذه العمليات بسهولة في أية بنى معطيات تدعم العمليات MINIMUM و DELETE و INSERT.

بل سنركز فقط على تخزين المفاتيح، وذلك لأننا سنركز على المفاتيح ولن نسمح بتخزين مفاتيح متكررة، فبدلاً من وصف عملية SEARCH، سننجز العملية الأبسط  $MEMBER(S, x)$ ، التي تعيد قيمةً منطقيةً تدل على وجود القيمة  $x$  حاليًا في المجموعة الديناميكية  $S$  أم لا.

استخدمنا حتى الآن المتوسط  $\equiv$  لفرضيتين متمازيتين: أولهما عدد العناصر في المجموعة الديناميكية، وثانيهما مجال القيم المحتملة. ولتجنب أي التباس آخر، سنستخدم من الآن فصاعدًا  $n$  للدلالة على عدد العناصر الموجودة حاليًا في المجموعة، و  $u$  للدلالة على مجال القيم المحتملة، وبذلك، فإن كلَّ عملية من عمليات أشجار van Emde Boas تنفذ بزمن  $O(\lg \lg u)$ . نسمي المجموعة  $\{0, 1, 2, \dots, u-1\}$  **عالم القيم universe of values** التي يمكن تخزينها، و  $u$  **حجم العالم universe size**. نفترض في هذا الفصل أن  $u$  هو من مضاعفات العدد 2، أي  $u = 2^k$  حيث  $k$  عدد صحيح أكبر أو يساوي الواحد.

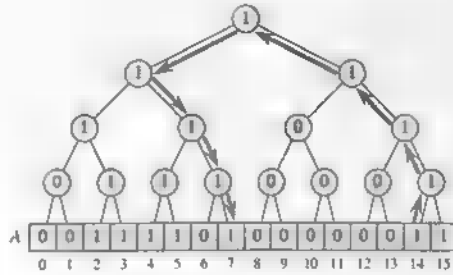
يبدأ المقطع 1.20 بمحصى بعض للمنهجيات البسيطة التي سنعملنا نسير في الاتجاه الصحيح. ثم نحسن هذه المنهجيات في المقطع 2.20، بتقديم بنى proto van Emde Boas، التي هي بنى عودية ولكنها لا تحقق غرضنا المتعلق بعمليات ذات زمن  $O(\lg \lg u)$ . وفي المقطع 3.20 نعدّل بنى proto-van Emde Boas لإنشاء أشجار Emde Boas  $\equiv$  ونبين كيفية تنحيز كلَّ عملية بزمن  $O(\lg \lg u)$ .

## 1.20 منهجيات مبدئية

سنفحص، في هذا المقطع، منهجيات متعددة لتخزين مجموعة ديناميكية. ومع أن أيًا من هذه المنهجيات لن نحقق في الزمن المرغوب  $O(\lg \lg u)$ ، فإننا سنحصل على أفكار تساعدنا على فهم أشجار van Emde Boas عندما نمرّ بنا لاحقًا في هذا الفصل.

### العنونة المباشرة

توفر العنونة المباشرة direct addressing - كما رأينا في المقطع 1-11 - أبسط منهجية لتخزين مجموعة ديناميكية. ولما كان اهتمامنا في هذا الفصل محصورًا في تخزين المفاتيح فقط، فيمكننا تبسيط منهجية العنونة المباشرة لتخزين المجموعة الديناميكية، وذلك باعتبارها شعاع بنات bit vector [انظر للنقطة في التمرين 2-1.11]. فلتخزين مجموعة ديناميكية من قيم العالم  $\{0, 1, 2, \dots, u-1\}$ ، نحفظ صيغة  $A[0..u-1]$  من  $u$  بتًا. حيث يأخذ العنصر  $A[x]$  القيمة 1 إذا كانت القيمة  $x$  ضمن المجموعة الديناميكية، والقيمة 0 في الحالة الأخرى. ومع أن بإمكاننا إجراء كلَّ من العمليات INSERT و DELETE و MEMBER بزمن  $O(1)$  باستخدام شعاع بنات، فإن كلاً من العمليات المتبقية - MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR - تستغرق زمنًا  $\Theta(u)$  في أسوأ الحالات، وذلك لأننا ربما نضطر إلى مسح  $\Theta(u)$  scan



**الشكل 1.20** شجرة ثنائية من بنات مُراكبة فوق شعاع بنات يمثل المجموعة {2, 3, 4, 5, 7, 14, 15} في حالة  $u = 16$ . تتضمن كل عقدة داخلية 1 إذا فقط إذا تضمنت ورقة ما في أشجارها الفرعية القيمة 1. وتبين الأسهم المسار التبع لتحديد العنصر السابق للقيمة 14 في المجموعة.

عنصرًا.<sup>2</sup> فعلى سبيل المثال، إذا تضمنت مجموعة ما القيمتين 0 و  $u - 1$  فقط، فقد نضطر - عند العثور على العنصر التالي للعنصر 0 - إلى شُح عناصر من 1 إلى  $u - 2$  قبل العثور على 1 في  $A[u - 1]$ .

### مراكبة بنية شجرة ثنائية في الأعلى

يمكننا اختصار عمليات المسح الطويلة لشعاع البنات بمراكبة شجرة ثنائية من البنات أعلى منه. يبين الشكل 1.20 مثالاً على ذلك. تكون عناصر شعاع البنات أوراق الشجرة الثنائية، وتتضمن كل عقدة داخلية القيمة 1 إذا فقط إذا تضمنت أية ورقة من شجرها الفرعية القيمة 1. بعبارة أخرى، فإن البت المخزن في عقدة داخلية هو نتيجة إجراء عملية "أو المنطقية" على ابنائها.

تستخدم العمليات - التي استقرت باستخدام شعاع بنات بسيط زمنًا  $O(u)$  في أسوأ الحالات - البنية الشجرية الآن:

- للعثور على القيمة الدنيا في المجموعة، ابدأ من الجذر واتجه نزولاً نحو الأوراق، بحيث تأخذ دومًا العقدة في أقصى اليسار التي تتضمن القيمة 1.
- للعثور على القيمة العظمى في المجموعة، ابدأ من الجذر واتجه نزولاً نحو الأوراق، بحيث تأخذ دومًا العقدة في أقصى اليمين التي تتضمن القيمة 1.

<sup>2</sup> نفترض في هذا الفصل أن MAXIMUM و MINIMUM تعيدان NIL إذا كانت المجموعة الديناميكية خالية، وأن SUCCESSOR و PREDECESSOR تعيدان NIL إذا لم يكن للعنصر للمعطى عنصر لاحق أو سابق على التوالي.

- للعثور على العنصر التالي successor ل  $x$ ، ابدأ من الورقة التي دليلها  $x$ ، واتجه صعودًا نحو الجذر حتى تدخل في عقدة من اليسار ويكون لهذه العقدة ابنًا أيمن  $z$  قيمته 1. ثم اتجه نزولًا عبر العقدة  $z$ ، بحيث تأخذ دومًا العقدة في أقصى اليسار التي تتضمن القيمة 1 (أي، اعثر على القيمة الدنيا في الشجرة الفرعية التي جذورها الابن الأيمن  $z$ ).
- للعثور على العنصر السابق predecessor ل  $x$ ، ابدأ من الورقة التي دليلها  $x$ ، واتجه صعودًا نحو الجذر حتى تدخل في عقدة من اليمين ويكون لهذه العقدة ابنًا أيسر  $z$  قيمته 1. ثم اتجه نزولًا عبر العقدة  $z$ ، بحيث تأخذ دومًا العقدة في أقصى اليسار التي تتضمن القيمة 1 (أي، اعثر على القيمة العظمى للشجرة الفرعية التي جذورها الابن الأيسر  $z$ ).

يبين الشكل 1.20 المسار المسلوک لإيجاد العنصر السابق 7 للقيمة 14.

نوسّع كذلك عمليتي INSERT و DELETE توسيعًا ملائمة. فنعد إدراج قيمة، نخزن القيمة 1 في كل عقدة موجودة على المسار البسيط الممتد من الورقة الموافقة وحتى الجذر. وعند حذف قيمة، نسرد انطلاقًا من الورقة الموافقة صعودًا باتجاه الجذر، بحيث نعيد حساب البت في كل عقدة داخلية من المسار على أنه نتيجة تطبيق "أو المنطقية" على ابنيّه.

ولما كان ارتفاع الشجرة هو  $\lg n$ ، وكانت كل عملية من العمليات السابقة تتطلب، على الأكثر، عبورًا واحدًا للشجرة باتجاه الأعلى، وعلى الأكثر، عبورًا آخر باتجاه الأسفل، فإن كل عملية تستغرق زمنًا  $O(\lg n)$  في أسوأ الحالات.

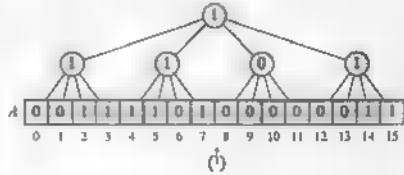
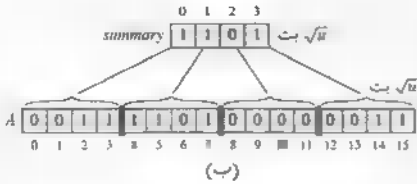
هذه المنهجية أفضل قليلًا فقط من استخدام شجرة حمراء-سوداء. حيث مازال بإمكاننا إنجاز عملية MEMBER بزمن  $O(1)$ ، في حين سيأخذ البحث في شجرة حمراء-سوداء زمنًا  $O(\lg n)$ . وهكذا نجد ثانية أنه إذا كان عدد العناصر  $n$  للمخزنة أصغر بكثير من حجم العالم  $n$ ، فستكون الشجرة الحمراء-السوداء أسرع في جميع العمليات الأخرى.

### مراقبة شجرة ذات ارتفاع ثابت

ما الذي يحدث إذا راكبنا شجرة ذات درجة أعلى؟ لنفترض أن حجم الفضاء هو  $u = 2^{2k}$ ، حيث  $k$  عدد صحيح، فيكون  $\sqrt{u}$  عددًا صحيحًا. فبدلاً من أن نراكب شجرة ثنائية فوق شعاع البتات، نراكب شجرة درجتها  $\sqrt{u}$ . يبين الشكل 2.20 (أ) شجرة مماثلة لشعاع البتات نفسه الذي في الشكل 1.20. إن ارتفاع الشجرة الناتجة هو 2 دومًا.

كما في السابق، نخزن كل عقدة داخلية نتيجة تطبيق "أو المنطقية" على البتات ضمن شجرتها الفرعية، بحيث تلخص العقد الـ  $\sqrt{u}$  الداخلية، التي عمقها 1، كل مجموعة من  $\sqrt{u}$  قيمة. وكما يبين الشكل 2.20 (ب)، يمكننا اعتبار هذه العقد صيغة  $summary[0..\sqrt{u}-1]$ ، حيث تتضمن





**الشكل 2.20** (أ) شجرة درجتها  $\sqrt{u}$  مراقبة فوق شعاع البتات الموجود في الشكل 1.20. تُخزّن كل عقدة داخلية قيمة "أو المنطقية" للبتات في الشجرة الفرعية. (ب) منظر للبتة نفسها عندما تعادل العقد الداخلية على العمق 1 باعتبارها صيغة  $summary[0..\sqrt{u}-1]$ ، حيث  $summary[i]$  هي قيمة "أو المنطقية" للصفيفة الجزئية  $A[i\sqrt{u}..(i+1)\sqrt{u}-1]$ .

القيمة  $summary[i]$  إذا وقطع إذا تضمنت الصفيفة الجزئية  $A[i\sqrt{u}..(i+1)\sqrt{u}-1]$  القيمة 1. نسمي هذه الصفيفة الجزئية من  $A$  ذات الـ  $\sqrt{u}$  بتًا **العنقود cluster** ذا الترتيب  $i$ . في حالة قيمة معطاة  $x$ ، يظهر البت  $A[x]$  في العنقود رقم  $\lfloor x/\sqrt{u} \rfloor$ . وهكذا أصبح الإجراء INSERT يستغرق الآن زمنًا  $O(1)$ : ولإدراج  $x$ ، ضع القيمة 1 في كلٍّ من  $A[x]$  و  $summary[\lfloor x/\sqrt{u} \rfloor]$ . يمكننا استخدام صيغة  $summary$  لإنجاز كلٍّ من العمليات: MINIMUM و MAXIMUM و SUCCESSOR و PREDECESSOR و DELETE بزمن  $O(\sqrt{u})$ :

- للعثور على القيمة الدنيا (العظمى)، ابحث عن العنصر الذي يتضمن 1 ويقع في أقصى يسار (يمين)  $summary$ ، وليكن  $summary[i]$ ، ثم ابحث خطيًا - ضمن العنقود ذي الترتيب  $i$  - عن القيمة 1 الموجودة في أقصى اليسار (اليمين).
- للعثور على العنصر التالي (السابق) للعنصر  $x$ ، ابحث أولاً باتجاه اليمين (اليسار) ضمن العنقود. فإذا وجدت القيمة 1، فيكون هذا الموقع هو النتيجة. وإلا، فاجعل  $i = \lfloor x/\sqrt{u} \rfloor$ ، وابحث باتجاه اليمين (اليسار) ضمن صفيفة  $summary$  ابتداءً من الدليل  $i$ . إن أول موقع يتضمن القيمة 1 يعطينا دليل عنقود. ابحث ضمن هذا العنقود عن أول 1 في أقصى اليسار (اليمين). هذا الموقع يحوي العنصر التالي (السابق).
- لحذف القيمة  $x$ ، اجعل  $i = \lfloor x/\sqrt{u} \rfloor$ . ضع القيمة 1 في  $A[x]$ ، ثم ضع في  $summary[i]$  قيمة ناتج "أو المنطقية" للبتات في العنقود ذي الترتيب  $i$ .

في كلٍّ من العمليات السابقة، نبحث، على الأكثر، في عنقودين من  $\sqrt{u}$  بتًا، إضافة إلى الصفيفة  $summary$ ، وهكذا فإن كل عملية تستغرق زمنًا  $O(\sqrt{u})$ .

يلدو، للهولة الأولى، وكأنتا أجريننا تعديلًا سليماً. أعطتنا مراكية شجرة ثنائية عمليات بزمن  $O(\lg u)$ ، والتي هي أسرع بالمقارنة من زمن  $O(\sqrt{u})$ . ولكن، سيتضح أن استخدام شجرة من درجة  $\sqrt{u}$  هي فكرة أساسية لأشجار van Emde Boas. ستابع هذا المسار في المقطع التالي.

## تعاريف

### 1-1.20

عدّل بنى للمعطيات في هذا المقطع لتدعم للفتاح للذاكرة.

### 2-1.20

عدّل بنى للمعطيات في هذا المقطع لتدعم للفتاح التي لها معطيات تابعة مرفقة.

### 3-1.20

لاحظ أن الطريقة التي يُجَدُّ فيها العنصر التالي والسابق لقيمة ما  $x$  - باستخدام البنى في هذا المقطع - لا تعتمد على كون  $x$  موجودة في المجموعة وقتئذٍ. بين كيف يمكنك العثور على العنصر التالي لـ  $x$  في شجرة بحث ثنائية عندما تكون  $x$  غير مخزنة في الشجرة.

### 4-1.20

افترض أنه بدلاً من مراكية شجرة درجتها  $\sqrt{u}$ ، راكمنا شجرة درجتها  $u^{1/k}$ ، حيث  $k$  ثابت أكبر من الواحد. ماذا سيكون ارتفاع هذه الشجرة؟ وكم ستستغرق كل عملية من العمليات؟

## 2.20 بنية عودية

نعدّل في هذا المقطع فكرة مراكية شجرة درجتها  $\sqrt{u}$  فوق شعاع بنات. فقد استخدمنا في المقطع السابق بنية مختصرة حجمها  $\sqrt{u}$ ، وكل عنصر فيها يشير إلى بنية أخرى حجمها  $\sqrt{u}$ . أما حالياً، فنجعل البنية عودية، مقلّصين حجم الفضاء إلى جذره، في كل مستوى من العودية. وابتداءً من فضاء حجمه  $u$ ، نجعل البنى تتضمن  $\sqrt{u} = u^{1/2}$  عنصراً، والتي تتضمن بدورها بنى من  $u^{1/4}$  عنصراً، والتي تتضمن بدورها بنى من  $u^{1/8}$  عنصراً، وهكذا نزولاً حتى الوصول إلى حجم أساسي هو 2.

نفترض للتبسيط، في هذا المقطع، أن  $u = 2^{2^k}$  حيث  $k$  عدد صحيح، وبحيث تكون  $u, u^{1/2}, u^{1/4}, \dots$  أعداداً صحيحة. قد يكون هذا القيد قاسياً عملياً، ويسمح أن تكون قيم  $u$  ضمن التسالية  $2, 4, 16, 256, 65536, \dots$  فقط. سنرى في المقطع التالي كيف نخفّف هذا القيد، ونفترض أن  $u = 2^{2^k}$  فقط حيث  $k$  عدد صحيح. ولما كانت البنية التي نفحصها في هذا المقطع هي بنية عمودية فقط لشجرة van Emde Boas الفعلية، فإننا نتساهل بخصوص هذا القيد للمساعدة على فهم المسألة.

وحيث إن هدفنا هو تحقيق أزمّة تنفيذ العمليات من رتبة  $O(\lg \lg u)$ ، فلنفكر في كيفية الحصول على أزمّة تنفيذ كهذه. كنا قد رأينا في نهاية المقطع 3.4 أنه بتغيير المتحولات يمكننا إثبات أن حل المعادلة التكرارية:

$$T(n) = 2T(\lfloor \sqrt{n} \rfloor) + \lg n \quad (1.20)$$

هو  $T(n) = O(\lg n \lg \lg n)$ . لنأخذ معادلة تكرارية مماثلة، ولكنها أبسط:

$$T(u) = T(\sqrt{u}) + O(1). \quad (2.20)$$

فإذا استخدمنا التقنية نفسها؛ أي تغيير المتحولات، أمكننا إثبات أن حل المعادلة التكرارية (2.20) هو  $T(u) = O(\lg \lg u)$ . ليكن  $m = \lg u$ ، وبحيث أن  $u = 2^m$  ويكون لدينا:

$$T(2^m) = T(2^{m/2}) + O(1).$$

وبتغيير الاسم  $S(m) = T(2^m)$  نحصل على المعادلة التكرارية الجديدة.

$$S(m) = S(m/2) + O(1).$$

وباستخدام الحالة 2 من الطريقة العامة master، يكون حل هذه المعادلة التكرارية هو  $S(m) = O(\lg m)$ .

نعيد تغيير الاسم من  $S(m)$  إلى  $T(u)$  فينتج  $T(u) = O(\lg m) = O(\lg \lg u)$ .

ستواجه المعادلة التكرارية 2.20 بحثاً عن بنية معطيات. لذا سنصمم بنية معطيات عودية تقلص بمقدار  $\sqrt{u}$  في كل مستوى من عوديتها. عندما نُغيّر عملية بنية المعطيات هذه، فإنها تستغرق زمناً ثابتاً في كل مستوى قبل أن تنتقل عودياً إلى المستوى الأدنى. حينئذٍ ستحدّد المعادلة التكرارية (2.20) زمن تنفيذ العملية.

فيما يلي طريقة أخرى للتفكير بكيفية الحصول على الحد  $\lg \lg u$  عند حل المعادلة التكرارية (2.20). عندما ننظر إلى حجم الفضاء في كل مستوى من بنية المعطيات العودية، نجد المتتالية  $u, u^{1/2}, u^{1/4}, u^{1/8}, \dots$ . فإذا أخذنا بالحسبان كمية البتات التي نحتاج إليها لتخزين حجم الفضاء في كل مستوى، فإننا نحتاج إلى  $\lg u$  في المستوى الأعلى، ويحتاج كل مستوى إلى نصف بتات المستوى السابق. وبوجه عام، إذا بدأنا بـ  $b$  بتاً وننصف عدد البتات في كل مستوى، سنحصل بعد  $\lg b$  مستوى على بت واحد فقط. ولما كانت  $b = \lg u$ ، فسيكون لدينا فضاء حجمه 2، بعد  $\lg \lg u$  مستوى.

وبالعودة إلى بنية المعطيات في الشكل 2.20، نجد أن قيمة معطاة  $x$  تقع في العقود رقم  $\lfloor x/\sqrt{u} \rfloor$ . فإذا كنا ننظر إلى  $x$  على أنه عدد صحيح ممثّل ثنائياً بـ  $\lg u$  بتاً، فإن رقم ذلك العقود،  $\lfloor x/\sqrt{u} \rfloor$ ، يعطى بالبتات  $(\lg u)/2$  الأكثر أهمية في  $x$ . ويظهر  $x$ ، ضمن هذا العقود، في الموقع  $x \bmod \sqrt{u}$ ، والذي يعطى بالبتات  $(\lg u)/2$  الأقل أهمية في  $x$ . ولما كنا بحاجة إلى الفهرسة بهذه الطريقة، فإننا نعرّف بعض الدوال التي تساعدنا على إجراء ذلك:

$$\text{high}(x) = \lfloor x/\sqrt{u} \rfloor.$$

$$\begin{aligned}\text{low}(x) &= x \bmod \sqrt{u}, \\ \text{index}(x, y) &= x/\sqrt{u} + y.\end{aligned}$$

تعطينا الدالة  $\text{high}(x)$  البتات  $(\lg u)/2$  الأكثر أهمية في  $x$ ، مولدة رقم عنقود  $x$ . وتعطينا الدالة  $\text{low}(x)$  البتات  $(\lg u)/2$  الأقل أهمية في  $x$ ، وتعطي موقع  $x$  ضمن عنقوده. أما الدالة  $\text{index}(x, y)$  فتبني رقم عنصر ابتداءً من  $x$  و  $y$ ، بحيث تعادل  $x$  على أنه البتات  $(\lg u)/2$  الأكثر أهمية في رقم العنصر، و تعادل  $y$  على أنه البتات  $(\lg u)/2$  الأقل أهمية. وتصبح لدينا المساواة  $x = \text{index}(\text{high}(x), \text{low}(x))$ .  $x$  ستكون قيمة  $u$  المستخدمة في كلٍّ من هذه الدوال هي دوماً حجم عالم بنية المعطيات التي نستدعي ضمنها الدالة، الذي سيتغير أثناء التزول في البنية العودية.

### 1.2.20 بني *proto van Emde Boas*

نصمّم، انطلاقاً من المعادلة التكرارية (2.20)، بنية معطيات عودية تدعم العمليات. ومع أن هذه البنية لن تحقق هدفاً في الوصول إلى زمن  $O(\lg \lg u)$  لبعض العمليات، فإنها ستُغلب باعتبارها أساساً لبنية شجرة *van Emde Boas* التي سترافها في المقطع 3.20.

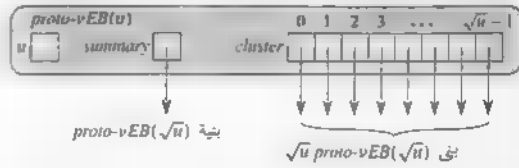
نعرف ضمن العالم  $\{0, 1, 2, \dots, u-1\}$  بنية *proto van Emde Boas*، أو بنية *proto-vEB* التي سنرمز لها بـ  $\text{proto-vEB}(u)$ ، عودياً كما يلي: تتضمن كلُّ بنية  $\text{proto-vEB}(u)$  واصفة  $u$  تحدد حجم عالمها. وهي تتضمن، إضافة إلى ذلك، ما يلي:

- إذا كان  $u = 2$ ، عندها يكون هو حجم الأساس، وتتضمن البنية صفيقة  $A[0..1]$  مؤلفة من بتين.
- وإلا، يكون  $u = 2^{2^k}$  حيث  $k \geq 1$  عدد صحيح، وبذلك يكون  $u \geq 4$ . تتضمن بنية للمعطيات  $\text{proto-vEB}(u)$ ، إضافة إلى حجم العالم  $u$ ، الواصفات التالية للبيئية في الشكل 3.20:

■ مؤشرًا إلى بنية  $\text{proto-vEB}(\sqrt{u})$  اسمه *summary*، و

■ صفيقة  $\text{cluster}[0.. \sqrt{u}-1]$  من  $\sqrt{u}$  مؤشرًا، يشر كلُّ منها إلى بنية  $\text{proto-vEB}(\sqrt{u})$ .

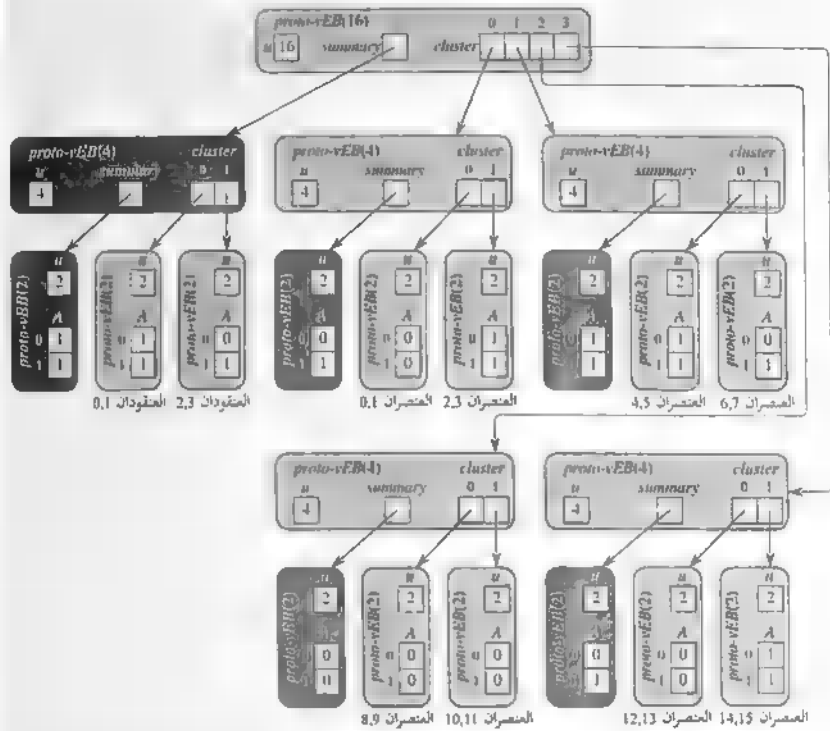
يُخزّن العنصر  $x$ ، حيث  $0 \leq x < u$ ، عودياً في العنقود رقم  $\text{high}(x)$  على أنه العنصر  $\text{low}(x)$  في ذلك العنقود. في البنية الثنائية للمستوى التي عرضناها في المقطع السابق، تُخزّن كلُّ عقدة صفيقة ملخص *summary* حجمها  $\sqrt{u}$ ، بحيث يتضمن كل عنصر فيها بتاً واحداً. وبمكنتنا، انطلاقاً من دليل كلِّ عنصر، حساب دليل البداية للصفيفة الجزئية التي حجمها  $\sqrt{u}$  والتي يلخصها البت. نستخدم في بنية  $\text{proto-vEB}$ ، مؤشرات صريحة بدلاً من حسابات الأدلة. تتضمن الصفيقة *summary* بتات الملخص التي تُخزّن عودياً في بنية  $\text{proto-vEB}$ ، أما الصفيقة *cluster* فتتضمن  $\sqrt{u}$  مؤشرًا.



**الشكل 3.20** المعلومات في بنية  $proto-vEB(u)$  عندما تكون  $u \geq 4$ . تتضمن البنية: حجم الفضاء  $u$  ومؤشرًا  $summary$  على بنية  $proto-vEB(\sqrt{u})$ ، وصيغة  $cluster[0..\sqrt{u}-1]$  من  $\sqrt{u}$  مؤشرًا على بنية  $proto-vEB(\sqrt{u})$ .

بين الشكل 4.20 بنية  $proto-vEB(16)$  موشعة تمامًا تمثل المجموعة  $\{2, 3, 4, 5, 7, 14, 15\}$ . فإذا كانت القيمة  $i$  موجودة في بنية  $proto-vEB$  التي يشير إليها  $summary$ ، فإن العقود ذات الترتيب  $i$  يتضمن قيمة ما في المجموعة الممثلة. وكما في الشجرة ذات الارتفاع الثابت، تمثل  $cluster[i]$  القيم من  $i\sqrt{u}$  إلى  $(i+1)\sqrt{u}-1$ ، التي تكون العقود ذات الترتيب  $i$ .

في المستوى الأساسي، تُخزن عناصر المجموعات الديناميكية الحالية في بعض بني  $proto-vEB(2)$ ، وتُخزن بقية بني  $proto-vEB(2)$  بنات الملخص. يظهر في الشكل - تحت كل من البنى الأساسية التي لا تكون ملخصًا - البنات التي تخزنها. فمثلاً، تُخزن بنية  $proto-vEB(2)$  التي عنوانها "العنصران 6 و 7" البت 6 في  $A[0]$ ، لأن العنصر 6 ليس من المجموعة، وتُخزن البت 7 في  $A[1]$ ، لأن العنصر 7 موجود في المجموعة. وكما في العناقد، فإن كل ملخص ليس سوى مجموعة ديناميكية حجم عالمها  $\sqrt{u}$ ، ولذلك نحن نعمل كل ملخص على أنه بنية  $proto-vEB(\sqrt{u})$ . أما بنات الملخص الأربعة في البنية  $proto-vEB(16)$  الأساسية، فموجودة في أقصى يسار بنية  $proto-vEB(4)$ ، وتظهر في النهاية في بني  $proto-vEB(2)$ . فمثلاً، في بنية  $proto-vEB(2)$  التي عنوانها "العنقدان 2، 3" لدينا  $A[0] = 0$  وهذا يدل على أن العقود 2 من البنية  $proto-vEB(16)$  (الذي يتضمن العناصر 8، 9، 10، 11) كلها أصفار، ولدينا  $A[1] = 1$ ، وهذا يدل على أن العقود 3 (الذي يتضمن العناصر 12، 13، 14، 15) فيه 1 واحد على الأقل. وتشير كل بنية  $proto-vEB(4)$  إلى خلاصتها، والتي هي نفسها مخزنة على أنها بنية  $proto-vEB(2)$ . انظر، على سبيل المثال، إلى بنية  $proto-vEB(2)$  الموجودة مباشرة إلى يسار تلك المسماة "العنصران 0، 1". لما كانت  $A[0]$  هي 0، فهذا يعني أن بنية "العنصران 0، 1" كلها أصفار، ولما كانت  $A[1]$  هي 1، فنحن نعلم أن بنية "العنصران 2، 3" تتضمن 1 واحد على الأقل.



**الشكل 4.20** بنية  $proto-vEB(16)$  تمثل المجموعة  $\{2,3,4,5,7,14,15\}$ . تشير البنية إلى أربع بني  $proto-vEB(4)$  في العقود  $cluster[0..3]$  وإلى بنية ملخص، التي هي أيضاً  $proto-vEB(4)$ . كل بنية  $proto-vEB(4)$  تشير إلى بنيتي  $proto-vEB(2)$  في عقود  $cluster[0..1]$  وإلى ملخص  $proto-vEB(2)$ . تتضمن كل بنية  $proto-vEB(2)$  صفيفة  $A[0..1]$  فقط مؤلفة من بتين. تخزن بني  $proto-vEB(2)$  للموضوعة فوق "العنصرين  $i$ ،  $j$ " البتتين  $i$  و  $j$  من المجموعة الديناميكية الحالية، وتخزن بني  $proto-vEB(2)$  للموضوعة فوق "العقودين  $i$ ،  $j$ " بتي الملخص للعقودين  $i$  و  $j$  في البنية ذات المستوى الأعلى  $proto-vEB(16)$ . تشير الظلال الغامقة - زيادةً في الإيضاح - إلى المستوى الأعلى من بنية  $proto-vEB$  التي تخزن معلومات الملخص لبنيها. وفيما عدا ذلك، تكون بنية  $proto-vEB$  هذه مطابقة لأية بنية  $proto-vEB$  أخرى لما حجم العالم ذاته.

## 2.2.20 العمليات على بنية $proto$ van Emde Boas

نصف الآن كيفية إنجاز العمليات على بنية  $proto-vEB$ . نفحص أولاً عمليات الاستعلام - MEMBER و MINIMUM و SUCCESSOR - التي لا تتغير بنية  $proto-vEB$ . ثم نقاش بعدها INSERT و DELETE.

وستترك **MAXIMUM** و **PREDECESSOR** - اللتين تناظران **MINIMUM** و **SUCCESSOR** على الترتيب - إلى التمرين 2.20-1.

تأخذ كلٌّ من العمليات: **MEMBER** و **SUCCESSOR** و **PREDECESSOR** و **INSERT** و **DELETE** موسطاً  $x$ ، إضافة إلى بنية  $V$  proto-VEB، حيث تفترض كلٌّ من هذه العمليات أن  $0 \leq x < V.u$ .

### كيف نحدّد وجود قيمة في المجموعة

لإنجاز **MEMBER(x)** نحتاج إلى العثور على البت الموافق لـ  $x$  ضمن بنية  $proto-VEB(2)$  المناسبة. ويمكننا إجراء ذلك بزم  $O(\lg \lg u)$ ، وذلك بالمرور على بنية **summary** جميعها. يأخذ الإجراء التالي بنية  $V$  proto-VEB وقيمة ما  $x$ ، ويعيد بتاً يدل على وجود  $x$  في المجموعة الديناميكية التي تمثلها  $V$ .

**PROTO-VEB-MEMBER(V, x)**

```

1  if  $V.u == 2$ 
2      return  $V.A[x]$ 
3  else return PROTO-VEB-MEMBER( $V.cluster[high(x)], low(x)$ )

```

يعمل الإجراء **PROTO-VEB-MEMBER** على النحو الآتي: يختار السطر 1 الحالة الأساسية، حيث  $V$  هي بنية  $proto-VEB(2)$ . ويعالج السطر 2 الحالة الأساسية، وذلك بإعادة البت المناسب من المصفوفة  $A$ . يتعامل السطر 3 مع الحالة العودية، "نزولاً" باتجاه أصغر بنية  $proto-VEB$  مناسبة. نبين القيمة  $high(x)$  أية بنية  $proto-VEB(\sqrt{u})$  نزر، وتحدّد  $low(x)$  أي عنصر ضمن بنية  $proto-VEB(\sqrt{u})$  نستعلم.

لننظر ماذا يحدث عندما نستدعي **PROTO-VEB-MEMBER(V, 6)** على بنية  $proto-VEB(16)$  في الشكل 4.20. لما كانت  $high(6) = 1$  عندما تكون  $u = 16$ ، فإننا نقوم بإجراء العودية ضمن بنية  $proto-VEB(4)$  في أعلى اليمين، ونسأل عن العنصر  $low(6) = 2$  في تلك البنية. في هذا الاستدعاء العودي يكون  $u = 4$ ، ولذلك نقوم بإجراء العودية مرة أخرى. ولما كانت  $u = 4$ ، فلدينا  $high(2) = 1$ ، و  $low(2) = 0$ ، ولذلك نسأل عن العنصر 0 من بنية  $proto-VEB(2)$  في أعلى اليمين. يتبيّن أن هذا الطلب العودي هو حالة أساسية، ولذلك يعيد  $A[0] = 0$  صعوداً عبر سلسلة الاستدعاءات العودية. وهكذا، نستنتج أن **PROTO-VEB-MEMBER(V, 6)** يعيد 0، وهذا يعني أن 6 ليس من المجموعة.

وبغية تحديد زمن تنفيذ **PROTO-VEB-MEMBER**، نرمز بـ  $T(u)$  إلى زمن تنفيذه على بنية  $proto-VEB(u)$ . إن كل استدعاء عودي يستغرق زمناً ثابتاً، لا يتضمن الزمن الذي تتطلبه الاستدعاءات العودية التي يقوم بها. عندما يقوم **PROTO-VEB-MEMBER** باستدعاء عودي، فإن الاستدعاء يكون على بنية  $proto-VEB(\sqrt{u})$ . وهكذا يمكننا توصيف زمن التنفيذ بالمعادلة التكرارية  $T(u) = T(\sqrt{u}) + O(1)$ ، التي شاهدناها آنفاً في للمعادلة التكرارية (2.20). وحلّ هذه للمعادلة هو  $T(u) = O(\lg \lg u)$ ، وبذلك

نستنتج أن  $\text{PROTO-VEB-MEMBER}$  يُنفَّذ بزمن  $O(\lg \lg u)$ .

### كيف نَجِدُ العنصر الأصغري

نبحث الآن في كيفية إنجاز عملية  $\text{MINIMUM}$ . يعيد الإجراء  $\text{PROTO-VEB-MINIMUM}(V)$  أصغر عنصر في بنية  $\text{proto-VEB } V$ ، أو يعيد  $\text{NIL}$  إذا كان  $V$  يمثل مجموعة خالية.

$\text{PROTO-VEB-MINIMUM}(V)$

```

1  if  $V.u == 2$ 
2      if  $V.A[0] == 1$ 
3          return 0
4      elseif  $V.A[1] == 1$ 
5          return 1
6      else return NIL
7  else  $\text{min-cluster} = \text{PROTO-VEB-MINIMUM}(V.summary)$ 
8      if  $\text{min-cluster} == \text{NIL}$ 
9          return NIL
10     else  $\text{offset} = \text{PROTO-VEB-MINIMUM}(V.cluster[\text{min-cluster}])$ 
11         return  $\text{index}(\text{min-cluster}, \text{offset})$ 
```

يُعمل هذا الإجراء كما يلي. يختار السطر 1 الحالة الأساسية، والتي تعالجها الأسطر 2-6 بقوة ضاربة  $\text{brute-force}$ . تعالج الأسطر 7-11 الحالة العودية. أولاً، يَجِدُ السطر 7 رقم أول عنقود يتضمن عنصراً من المجموعة. يقوم بذلك باستدعاء  $\text{PROTO-VEB-MINIMUM}$  عودياً على  $V.summary$ ، وهو بنية  $\text{proto-VEB}(\sqrt{u})$ . يُسَيِّدُ السطر 7 رقم العنقود هذا إلى  $\text{min-cluster}$  للتحويل إلى  $\text{min-cluster}$ . فإذا كانت المجموعة خالية، يعيد الاستدعاء العودي القيمة  $\text{NIL}$ ، ويعيد السطر 8 القيمة  $\text{NIL}$ . وإلا، يكون أصغر عنصر في المجموعة موجوداً في مكان ما في العنقود رقم  $\text{min-cluster}$ . يَجِدُ الاستدعاء العودي في السطر 10 الانزياح  $\text{offset}$  ضمن عنقود أصغر عنصر في هذا العنقود. وفي النهاية، يبي السطر 11 قيمة أصغر عنصر انطلاقاً من رقم العنقود والانزياح، ويعيد هذه القيمة.

ومع أن الاستعلام ضمن معلومة للمخصص يسمح بالعثور على العنقود الذي يتضمن أصغر عنصر بسرعة، إلا أن هذا الإجراء لا يُنفَّذ بزمن  $O(\lg \lg u)$  في أسوأ الحالات، لأنه يقوم باستدعاءين عوديين على بنية  $\text{proto-VEB}(\sqrt{u})$ . نرسم إلى زمن تنفيذ  $\text{PROTO-VEB-MINIMUM}$  في أسوأ الحالات، على بنية  $\text{proto-VEB}(u)$ ، بالرمز  $T(u)$ ، فتكون لدينا المعادلة التكرارية:

$$T(u) = 2T(\sqrt{u}) + O(1). \quad (3.20)$$

نستخدم، مرة أخرى، تغيير التحويلات لحل هذه المعادلة، حيث نجعل  $m = \lg u$ ، وهذا يعطي

$$T(2^m) = 2T(2^{m/2}) + O(1).$$



ونتسمية  $S(m) = T(2^m)$  نحصل على

$$S(m) = 2S(m/2) + O(1) ,$$

التي حلها، وفق الحالة 1 من الطريقة العامة، هو  $S(m) = \Theta(m)$ . وبإعادة تغيير  $S(m)$  إلى  $T(u)$ ، يكون لدينا  $T(u) = T(2^m) = S(m) = \Theta(m) = \Theta(\lg u)$ ، وبذلك، نرى أنه بسبب الاستدعاء العودي الثاني بنقطة PROTO-VEB-MINIMUM بزم  $\Theta(\lg u)$ ، وليس بالزمن المطلوب  $O(\lg \lg u)$ .

### كيف نجد العنصر التالي

إن عملية SUCCESSOR هي أشد سوءاً مما سبقها. في أسوأ الحالات، يقوم الإجراء باستدعاءين عوديين، إضافة إلى استدعاء PROTO-VEB-MINIMUM. ويعيد الإجراء  $\text{PROTO-VEB-MINIMUM}(V, x)$  أصغر العناصر في بنية proto-VEB  $V$  الذي هو أكبر من  $x$ ، أو يعيد NIL إذا لم يوجد عنصر في  $V$  أكبر من  $x$ . لا يتطلب هذا الإجراء أن يكون  $x$  عنصراً member في المجموعة، ولكنه يفترض أن يكون  $0 \leq x < V$ .

PROTO-VEB-SUCCESSOR( $V, x$ )

```

1  if  $V.u == 2$ 
2      if  $u == 0$  and  $V.A[1] == 1$ 
3          return 1
4      else return NIL
5  else  $offset = \text{PROTO-VEB-SUCCESSOR}(V.cluster[high(x)], low(x))$ 
6      if  $offset \neq \text{NIL}$ 
7          return  $index(high(x), offset)$ 
8      else  $succ-cluster = \text{PROTO-VEB-SUCCESSOR}(V.summary, high(x))$ 
9          if  $succ-cluster == \text{NIL}$ 
10             return NIL
11         else  $offset = \text{PROTO-VEB-MINIMUM}(V.cluster[succ-cluster])$ 
12         return  $index(succ-cluster, offset)$ 
```

يعمل الإجراء PROTO-VEB-SUCCESSOR على النحو الآتي: يختار السطر 1، كالعادة، الحالة الأساسية، التي تعالجها الأسطر 4-2 بالقوة الضاربة: الطريقة الوحيدة التي يمكن أن يكون فيها للعنصر  $x$  عنصرٌ تالي ضمن بنية  $\text{proto-VEB}(2)$  هي عندما تكون  $x = 0$  وتكون قيمة  $A[1]$  هي 1. تعالج الأسطر 5-12 الحالة العودية. يبحث السطر 5 عن عنصرٍ تالي لـ  $x$  ضمن عقوده  $x$ ، مستنداً النتيجة إلى  $offset$ . يُحدّد السطر 6 وجود عنصرٍ تالي لـ  $x$  ضمن عقوده؛ فإذا كان له عنصرٌ تالي، يحسب السطر 7 قيمة هذا العنصر ويعيدها. وإلا، علينا أن نبحث في عنايق أخرى. يستد السطر 8 رقم العقود التالي غير الحالي إلى  $succ-cluster$ ، مستخدماً معلومات المخصص لإيجاده. يختار السطر 9 مطابقة قيمة  $succ-cluster$  إلى  $cluster$  لـ NIL، ويعيد السطر 10 القيمة NIL إذا كانت العنايق التالية جميعها خالية. إذا لم تكن قيمة

*succ-cluster* هي *nil*، يُستد السطر 11 إلى *offset* أول عنصر ضمن هذا العقود، وبحسب السطر 12 أصغر عنصر في هذا العقود ويعيله.

في أسوأ الحالات، يستدعي *PROTO-VEB-SUCCESSOR* نفسه عودياً مرتين على بنى *proto-veb(√u)*، ويستدعي *PROTO-VEB-MINIMUM* مرة واحدة على بنى *proto-veb(√u)*. وبذلك، تكون المعادلة التكرارية الخاصة بزمن تنفيذ *PROTO-VEB-SUCCESSOR* بأسوأ الحالات هي  $T(u)$ :

$$\begin{aligned} T(u) &= 2T(\sqrt{u}) + \Theta(\lg \sqrt{u}) \\ &= 2T(\sqrt{u}) + \Theta(\lg u) . \end{aligned}$$

يمكننا استخدام التقنية نفسها التي استعملناها في المعادلة التكرارية (1.20) لنبين أن حل هذه المعادلة التكرارية هو  $T(u) = \Theta(\lg u \lg \lg u)$ . وهكذا، فإن *PROTO-VEB-SUCCESSOR* أبداً تقاربياً من *PROTO-VEB-MINIMUM*.

### إدراج عنصر

لإدراج عنصر، نحن بحاجة إلى أن يكون إدراجه في العقود للماتم وإلى وضع القيمة 1 في بت الملخص لهذا العقود. يُدرج الإجراء  $PROTO-VEB-INSERT(V, x)$  القيمة  $x$  في بنى *proto-veb*.

*PROTO-VEB-INSERT(V, x)*

- 1 if  $V.u = 2$
- 2      $V.A[x] = 1$
- 3 else *PROTO-VEB-INSERT*( $V.cluster[high(x)], low(x)$ )
- 4     *PROTO-VEB-INSERT*( $V.summary, high(x)$ )

في الحالة الأساسية، يضع السطر 2 القيمة 1 في البت للماتم في الصفيفة  $A$ . في الحالة العودية، يُدرج الاستدعاء العودي، في السطر 3،  $x$  في العقود للماتم، ويضع السطر 4 القيمة 1 في بت الملخص لهذا العقود. ولما كان الإجراء *PROTO-VEB-INSERT* يقوم باستدعاءين عوديين في أسوأ الحالات، فإن المعادلة التكرارية (3.20) تُصِف زمن تنفيذ هذا الإجراء. لذلك، يُقد *PROTO-VEB-INSERT* بزمن  $\Theta(\lg u)$ .

### حذف عنصر

إن عملية *DELETE* أعقد من الإدراج. لأنه إذا كان بإمكاننا دوماً وضع القيمة 1 في بت الملخص عند الإدراج، فإننا لا نستطيع دوماً إعادة وضع القيمة 0 في بت الملخص نفسه عند الحذف. ونحن بحاجة إلى تحديد: هل تساوي قيمة أحد البتات في العقود للماتم القيمة 1؟ حسب تعريفنا لبنى *proto-veb*، علينا أن نفحص جميع البتات التي عددها  $\sqrt{u}$  ضمن عقود لنحدد: هل يساوي أحدها القيمة 1؟ ثم حلّ بدل، وهو أن نضيف واصفة  $\pi$  إلى بنى *proto-veb*، ونُغذ عدد العناصر في البنية. سنترك تنحيز

PROTO-VEB-DELETE إلى التمرينين 2-2.20 و 3-2.20.

من الواضح أن علينا تعديل بنية proto-VEB لتخفيض كل عملية بحيث يكون فيها استدعاء عودي واحد على الأكثر. سترى في المقطع التالي كيف يجري ذلك.

### تمارين

#### 1-2.20

اكتب شبه رماز للإجراءين PROTO-VEB-MAXIMUM و PROTO-VEB-PREDECESSOR.

#### 2-2.20

اكتب شبه رماز للإجراء PROTO-VEB-DELETE. يجب أن محدّث يتّ للمخصص الملائم بمسح البتات المرتبطة ضمن العنقود. ما هو زمن التنفيذ في أسوأ الحالات لإجرائك؟

#### 3-2.20

أضف الوصفة  $n$  إلى كل بنية proto-VEB، التي تعطي عدد العناصر الموجودة حاليًا في المجموعة التي تمثّلها، واكتب شبه رماز للإجراء PROTO-VEB-DELETE الذي يستخدم الوصفة  $n$  ليحدد متى يضع القيمة 0 في بتات المخصص. ما هو زمن تنفيذ الحالة الأسوأ لإجرائك؟ ما هي الإجراءات الأخرى التي تحتاج إلى تغيير بسبب هذه الوصفة الجديدة؟ هل تؤثر هذه التغييرات على أزمان تنفيذها؟

#### 4-2.20

عدّل بنية proto-VEB لتدعم المفاتيح المكررة.

#### 5-2.20

عدّل بنية proto-VEB لتدعم المفاتيح التي لها معطيات تابعة مرفقة.

#### 6-2.20

اكتب شبه رماز لإجراء ينشئ بنية  $proto-VEB(u)$ .

#### 7-2.20

أثبت أنه إذا تمّ السطر 9 من PROTO-VEB-MINIMUM، فإن بنية proto-VEB تكون خالية.

#### 8-2.20

افترض أننا صممنا بنية proto-VEB بحيث أن كل صيغة cluster فيها تتضمن  $u^{1/4}$  عنصرًا فقط. ماذا ستكون أزمان تنفيذ كلٍّ من هذه العمليات؟

### 3.20 شجرة van Emde Boas

إن بنية proto-vEB المعروضة في المقطع السابق قريبة لما نرغب بتحقيقه من حيث أزمان تنفيذ  $O(\lg \lg u)$ . لكنها لا تفي بالفرض، لأننا يجب أن نقوم بالكثير من الاستدعاءات العودية في معظم العمليات. سنصمم في هذا المقطع بنية معطيات شبيهة ببنية proto-vEB لكنها تخزن معلومات أقل بقليل، وبذلك تنفي الحاجة إلى بعض الاستدعاءات العودية.

لاحظنا في المقطع 2.20، أن الفرضية التي وضعناها بخصوص حجم العالم - أي  $u = 2^{2^k}$ ، حيث  $k$  عدد صحيح - مقيدة جدًا، وهذا يجعل القيم الممكنة لـ  $u$  مجموعة محدودة جدًا. لذلك سنسمح، بدءًا من الآن، بأن يكون حجم العالم  $u$  أية قوة صحيحة من 2، وعندما لا تكون  $\sqrt{u}$  عددًا صحيحًا - أي، إذا كانت  $u$  قوة فردية لـ 2 ( $u = 2^{2k+1}$ )، حيث  $k \geq 0$  عدد صحيح) - فإننا سنقسم البتات التي عددها  $\lg u$  لعدد ما إلى البتات  $\lceil \lg u \rceil / 2$  الأكثر أهمية، والبتات  $\lfloor \lg u \rfloor / 2$  الأقل أهمية. والتبسيط، نرمز لـ  $\lceil \lg u \rceil / 2$  (أي "الجزء التربيعي الأعلى" لـ  $u$ ) بالرمز  $\sqrt{u}$  ولـ  $\lfloor \lg u \rfloor / 2$  (أي "الجزء التربيعي الأدنى" لـ  $u$ ) بالرمز  $\sqrt[2]{u}$ ، فيكون لدينا  $u = \sqrt{u} \cdot \sqrt[2]{u}$ ، وعندما يكون  $u$  قوة زوجية لـ 2 ( $u = 2^{2k}$ )، حيث  $k$  عدد صحيح، يكون لدينا  $\sqrt{u} = \sqrt[2]{u} = \sqrt{u}$ . وحيث إننا سمحنا بأن تكون  $u$  قوة فردية لـ 2، فيجب إعادة تعريف دوالنا المساعدة المذكورة في المقطع 2.20:

$$\text{high}(x) = \lfloor x / \sqrt{u} \rfloor,$$

$$\text{low}(x) = x \bmod \sqrt{u},$$

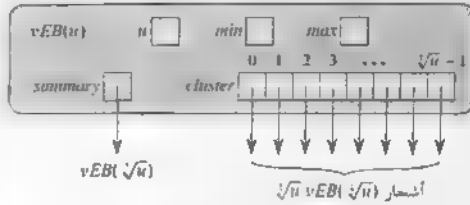
$$\text{index}(x, y) = x \sqrt{u} + y.$$

#### 1.3.20 أشجار van Emde Boas

تعدّل شجرة van Emde Boas، أو شجرة vEB، بنية proto-vEB. نرمز لشجرة vEB حجم عالمها  $u$  بالرمز  $vEB(u)$ ، وما لم تكن  $u$  مساوية الحجم الأساسي 2، فإن الوصفة summary تشير إلى شجرة  $vEB(\sqrt{u})$ ، وتشير الصيغة  $cluster[0.. \sqrt{u} - 1]$  إلى  $\sqrt{u} \cdot vEB(\sqrt[2]{u})$  شجرة. وكما هو موضح في الشكل 5.20، تتضمن شجرة vEB واصفتين غير موجودتين في بنية proto-vEB:

- تخزن  $\min$  أصغر عناصر شجرة vEB، و
- تخزن  $\max$  أكبر عناصر شجرة vEB.

إضافة إلى ذلك، لا يظهر العنصر المخزن في  $\min$  في أيٍّ من الأشجار العودية التي عددها  $vEB(\sqrt{u})$  التي تشير إليها الصيغة cluster. لذلك، فإن عدد العناصر المخزنة في شجرة  $vEB(u)$  هو  $V \cdot \min$  إضافة إلى جميع العناصر المخزنة عوديًا في الأشجار التي عددها  $vEB(\sqrt{u})$  والتي يشير إليها



**الشكل 5.20** المعلومات في شجرة  $vEB(u)$  عندما تكون  $u > 2$ . تتضمن البنية حجم العالم  $u$ ، والعنصرين  $min$  و  $max$ ، ومؤشرًا  $summary$  على شجرة  $vEB(\sqrt{u})$ ، وصيغة  $cluster[0..\sqrt{u}-1]$  من  $\sqrt{u}$  مؤشرًا على شجرة  $vEB(\sqrt{u})$ .

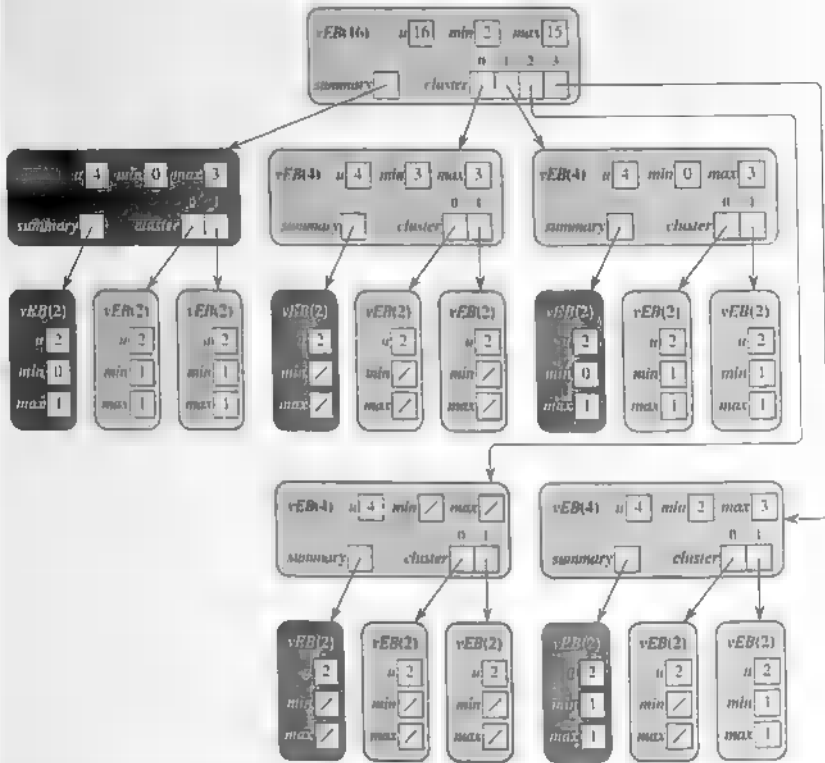
$V.cluster[0..\sqrt{u}-1]$ . لاحظ أننا نعامل  $min$  و  $max$  معاملًا مختلفة عندما تتضمن شجرة  $vEB$  عنصرين أو أكثر: العنصر المخزن في  $min$  لا يظهر في أيٍّ من العناقد، في حين يظهر العنصر المخزن في  $max$  في

ولما كان الحجم الأساسي هو 2، فإن شجرة  $vEB(2)$  لا تحتاج إلى الصيغة  $A$  التي كانت لدى بنية  $proto-vEB(2)$  الموافقة. وبدلاً من ذلك، يمكننا تحديد عناصرها من واصفتيها  $min$  و  $max$ . فإذا كانت شجرة  $vEB$  خالية من العناصر - بصرف النظر عن حجم عالمها  $u$  - فإن قيمة كلٍّ من  $min$  و  $max$  هي NIL.

يبين الشكل 6.20 شجرة  $vEB(16)$  تتضمن المجموعة  $\{2, 3, 4, 5, 7, 14, 15\}$ . ولما كان أصغر العناصر هو 2، فإن  $V.min$  تساوي 2، وحتى لو كان  $high(2) = 0$ ، فإن العنصر 2 لا يظهر في الشجرة  $vEB(4)$  التي يشار إليها  $V.cluster[0]$ : لاحظ أن  $V.cluster[0].min$  يساوي 3، وبهذا لا يكون العنصر 2 في شجرة  $vEB$ . وبالمثل، لما كانت قيمة  $V.cluster[0].min$  تساوي 3، وكان 2 و 3 هما العنصران الوحيدين في  $V.cluster[0]$ ، فإن العقودين  $vEB(2)$  ضمن  $V.cluster[0]$  خاليان.

سنبين لاحقاً أن الواصفتين  $min$  و  $max$  هما دورٌ أساسيٌّ في تخفيض عدد الاستدعاءات العودية ضمن العمليات على أشجار  $vEB$ . سنساعدنا هاتان الواصفتان على أربعة صعد:

1. لم تعد العمليتان MINIMUM و MAXIMUM بحاجة إلى عودية، لأن بإمكانهما إعادة قيم  $min$  و  $max$  فقط.
2. يمكن أن تجنب العملية SUCCESSOR إجراء استدعاءٍ عوديٍّ لتحديد: هل العنصر التالي لقيمة ما  $x$  موجودٌ ضمن  $high(x)$ ؟ وذلك لأن العنصر التالي لـ  $x$  يقع ضمن عقودها إذا وفقط إذا كانت  $x$  أصغر تماماً من الوصفة  $max$  لعقودها. يمكن تطبيق برهان مشابه على PREDECESSOR و  $min$ .



**الشكل 6.20** شجرة  $vEB(16)$  متوافقة لشجرة proto-vEB الموضوعة في الشكل 4.20. تخزن هذه الشجرة المجموعة {2, 3, 4, 5, 7, 14, 15}. ندل '/' على قيم NIL. لا تظهر القيمة المخزنة في الوصفة min لشجرة vEB في أي من عناقيدها. استعملت الظلال الغامقة هنا لنفس الغرض المذكور في الشكل 4.20.

3. يمكننا بزمّن ثابت معرفة: هل الشجرة vEB عالية، أم أنها تحتوي على عنصر وحيد، أم أنها تحتوي على عنصرين على الأقل؟ وذلك بالاعتماد على قيمتي  $min$  و  $max$ . ومنستفيد من هذه الإمكانية في عمليتي INSERT و DELETE. فإذا لم تكن  $min$  و  $max$  تساويان NIL، فإن شجرة vEB تكون عالية من العناصر. وإذا كانت  $min$  و  $max$  لا تساويان NIL، ولكنهما متساويان، فإن شجرة vEB تتضمن عنصرًا واحدًا غامًا. وإذا كانت  $min$  و  $max$  لا تساويان NIL، ولكنهما غير متساويين، فإن شجرة vEB تتضمن عنصرين أو أكثر.

4. إذا علمنا أن شجرة vEB خالية، يمكننا إدراج عنصر فيها بتعديل واصفيتها  $min$  و  $max$  فقط. ومن ثمّ يمكننا الإدراج في شجرة vEB خالية بزمّن ثابت. وبالمثل، إذا علمنا أن شجرة vEB فيها عنصر واحد فقط، يمكننا أن نحذف ذلك العنصر بزمّن ثابت بتعديل  $min$  و  $max$  فقط. ستتمكننا هذه الخواص من تخفيض سلسلة الاستدعاءات العودية.

فإذا كان حجم العالم  $u$  قوةً فرديةً للعدد 2، فإن الفرق بين حجمي شجرة vEB الملتخص والعناقد لن يؤثر في أزمنة التنفيذ التقاربية لعمليات شجرة vEB. وستكون جميع أزمنة تنفيذ الإجراءات العودية التي تنجز عمليات شجرة vEB موصّفة بالمعادلة التكرارية.

$$T(u) \leq T(\sqrt[3]{u}) + O(1) . \quad (4.20)$$

تشبه هذه للمعادلة التكرارية للمعادلة (2.20)، وستحلها بطريقة مشابهة. ليكن لدينا  $m = \lg u$ ، نعيد كتابة المعادلة التكرارية بالصيغة:

$$T(2^m) \leq T(2^{\lfloor m/2 \rfloor}) + O(1) .$$

وملاحظة أن  $\lfloor m/2 \rfloor \leq 2m/3$  لجميع قيم  $m \geq 1$ ، يكون لدينا:

$$T(2^m) \leq T(2^{2m/3}) + O(1) .$$

وبافتراض  $S(m) = T(2^m)$ ، نعيد كتابة المعادلة التكرارية الأخيرة هذه بالصيغة:

$$S(m) \leq S(2m/3) + O(1) ,$$

وحلّها - وفق الحالة 2 من الطريقة الأساسية - هو  $S(m) = O(\lg m)$ . (ليس هناك فارق - في الحل التقاربي - بين الكسرين  $2/3$  و  $1/2$ ، لأننا عندما نطبق الطريقة الأساسية، نجد أن  $\log_{3/2} 1 = \log_2 1 = 0$ ). وبذلك، يكون لدينا  $T(u) = T(2^m) = S(m) = O(\lg m) = O(\lg \lg u)$ .

قبل استخدام شجرة van Emde Boas، علينا معرفة حجم العالم  $u$ ، بحيث نستطيع إنشاء شجرة van Emde Boas بالحجم اللازم الذي يمثّل في البداية مجموعةً خالية. يُطلب في المسألة 1-20 برهان أن الحجم الكلي المطلوب لشجرة van Emde Boas هو  $O(u)$ ، وأنه يمكن إنشاء شجرة خالية مباشرة بزمّن  $\Theta(u)$ . وبالمقابل، يمكننا إنشاء شجرة حمراء-سوداء خالية بزمّن ثابت. لذلك، ربما لا نرغب في استخدام شجرة van Emde Boas عندما نجرى عددًا صغيرًا فقط من العمليات، لأن زمن إنشاء بنية المعطيات قد يتجاوز الزمن الذي نكسبه في العمليات المنفصلة. هذه السبب ليست هامة، لأننا نستخدم عادةً بنيةً معطيات بسيطةً، مثل صفيفة أو قائمة مترابطة، لتمثيل مجموعة عددٍ عناصرها قليل.

### 2.3.20 العمليات على شجرة van Emde Boas

نحن الآن جاهزون لمعرفة كيفية إنجاز العمليات على شجرة van Emde Boas. سندرس، كما فعلنا في بنية proto van Emde Boas، عمليات الاستعلامات أولاً، ثم INSERT و DELETE. ونظرًا لعدم التناظر الطفيف

بين العنصرين الأصغر والأعظم في شجرة  $vEB$  - عندما تتضمن شجرة  $vEB$  عنصرين على الأقل، لا يظهر العنصر الأصغر ضمن عنقود، في حين يظهر العنصر الأعظم - سنورد شبه رماز لجميع عمليات الاستعلام الخمس. وكما في العمليات على بني  $proto\ van\ Emde\ Boas$ ، نفترض العمليات التي تأخذ موسطين  $V$  و  $x$ ، أن  $0 \leq x < V.u$ ، حيث  $V$  هي شجرة  $van\ Emde\ Boas$  و  $x$  هو عنصر.

### كيف نجد العنصرين الأصغري والأعظمي

لما كنا نغزّن العنصرين الأصغري والأعظمي في الواصفتين  $min$  و  $max$ ، فهناك عملتان مؤلفتان من سطر واحد، تأخذان زمناً ثابتاً:

$vEB-TREE-MINIMUM(V)$

1 return  $V.min$

$vEB-TREE-MAXIMUM(V)$

1 return  $V.max$

### تحديث وجود قيمة ما في المجموعة

للإجراء  $vEB-TREE-MEMBER(V, x)$  حالة عودية كذلك الموجودة في  $PROTO-vEB-MEMBER$ ، لكن الحالة الأساسية مختلفة قليلاً. لذا فإننا نفحص مباشرة للمساواة بين  $x$  والعنصر الأصغري أو الأعظمي. وحيث إن شجرة  $vEB$  لا تغزّن البتات مثلما تفعل بنية  $proto-vEB$ ، فإننا نصمّم  $vEB-TREE-MEMBER$  للحصول على  $TRUE$  أو  $FALSE$  بدلاً من 1 أو 0.

$vEB-TREE-MEMBER(V, x)$

1 if  $x == V.min$  or  $x == V.max$

2 return  $TRUE$

3 elseif  $V.u == 2$

4 return  $FALSE$

5 else return  $vEB-TREE-MEMBER(V.cluster[high(x)], low(x))$

يفحص السطر 1 المساواة بين  $x$  والعنصر الأصغري أو الأعظمي. فإذا تحققت، فإن السطر 2 يعيد القيمة  $TRUE$ . وإلا، يختار السطر 3 الحالة الأساسية. ولما كانت شجرة  $vEB(2)$  لا تتضمن سوى العناصر الموجودة في  $min$  و  $max$ ، إذا كانت هذه هي الحالة الأساسية، فإن السطر 4 يُعيد القيمة  $FALSE$ . تجري معالجة الاحتمال الآخر - إذا لم تكن هذه هي الحالة الأساسية، وكانت  $x$  لا تساوي  $min$  ولا  $max$  - بالاستدعاء العودي في السطر 5.

نوصّف المعادلة التكرارية (4.20) زمن تنفيذ إجراء  $vEB-TREE-MEMBER$ ، لذا فإنه يستغرق

زمنًا  $O(\lg u)$ .



### كيف نجدُ اللاحق والسابق

نوضح فيما يلي كيف ننجز عملية `SUCCESSOR`. تذكر أن إجراء `PROTO-vEB-SUCCESSOR(V, x)` قد يُجري استدعاءين عوديين: أحدهما لتحديد: هل للاحق  $x$  موجود في عنقود  $x$  نفسه؟ وإذا لم يكن كذلك، فآخر للعنود على العنقود الذي يتضمن للاحق  $x$ . ولما كان باستطاعتنا الوصول سريعاً إلى القيمة العظمى في شجرة `vEB`، فيمكننا تجنب إجراء استدعاءين عوديين، والقيام بدلاً من ذلك باستدعاء عودي واحد على عنقود أو على الملخص، ولكن ليس عليهما معاً.

`vEB-TREE-SUCCESSOR(V, x)`

```

1  if  $V.u == 2$ 
2      if  $x == 0$  and  $V.max == 1$ 
3          return 1
4      else return NIL
5  elseif  $V.min \neq \text{NIL}$  and  $x < V.min$ 
6      return  $V.min$ 
7  else  $max-low = \text{vEB-TREE-MAXIMUM}(V.cluster[high(x)])$ 
8      if  $max-low \neq \text{NIL}$  and  $low(x) < max-low$ 
9           $offset = \text{vEB-TREE-SUCCESSOR}(V.cluster[high(x)], low(x))$ 
10         return  $index(high(x), offset)$ 
11     else  $succ-cluster = \text{vEB-TREE-SUCCESSOR}(V.summary, high(x))$ 
12         if  $succ-cluster == \text{NIL}$ 
13             return NIL
14         else  $offset = \text{vEB-TREE-MINIMUM}(V.cluster[succ-cluster])$ 
15         return  $index(succ-cluster, offset)$ 
    
```

لهذا الإجراء ستة تعليمات `return`، وحالات متعددة. نبدأ بالحالة الأساسية في الأسطر 2-4، التي تعيد القيمة 1 في السطر 3 إذا كنا نحاول إيجاد للاحق 0 وكان 1 موجوداً في المجموعة المؤلفة من عنصرين؛ وإلا، تعيد الحالة الأساسية القيمة `NIL` في السطر 4.

إذا لم تكن في الحالة الأساسية، فإننا نتفقد بعد ذلك في السطر 5: هل  $x$  أصغر تماماً من أصغر عنصر؟ فإذا كان كذلك، تعيد ببساطة أصغر عنصر في السطر 6.

إذا وصلنا إلى السطر 7، نعرف عندها أننا لسنا في الحالة الأساسية، وأن  $x$  أكبر أو يساوي أصغر قيمة في شجرة `vEB`. يسند السطر 7 قيمة أعظم عنصر في عنقود  $x$  إلى `max-low`. إذا كان عنقود  $x$  يتضمن عنصراً أكبر من  $x$ ، نعلم عندها أن للاحق  $x$  موجود في مكان ما ضمن عنقود  $x$ . يُختبر السطر 8 هذا الشرط. إذا كان للاحق  $x$  موجوداً في عنقود  $x$ ، يحدّد السطر 9 مكانه في العنقود، ويعيد السطر 10 العنصر اللاحق بنفس طريقة السطر 7 من `PROTO-vEB-SUCCESSOR`.

نصل إلى السطر 11 إذا كان  $x$  أكبر أو يساوي أعظم عنصر في عنقوده. في هذه الحالة، نُجَدِّد الأسطر 11-15 لاحقاً  $x$  بنفس الطريقة للمروضة في السطور 8-12 من **PROTO-VEB-SUCCESSOR**.

من السهل رؤية كيف توصف للمعادلة التكرارية (4.20) زمن تنفيذ **VEB-TREE-SUCCESSOR**. واعتماداً على نتيجة الاختبار في السطر 8، يستدعي الإجراء نفسه عودياً إما في السطر 9 (على شجرة **veb** وبحجم عالمي  $\sqrt{u}$ ) وإما في السطر 11 (على شجرة **veb** بحجم عالمي  $\sqrt{u}$ ). في كلتا الحالتين، يكون الاستدعاء العودي الوحيد هو على شجرة **veb** وبحجم عالمي  $\sqrt{u}$  على الأكثر. يستغرق باقي الإجراء زمناً  $O(1)$ ، ومن ضمه الاستدعاءات **veb-TREE-MINIMUM** و **veb-TREE-MAXIMUM**. لذلك، يُنفَّذ **VEB-TREE-SUCCESSOR** بزمن  $O(\lg \lg u)$  في أسوأ الحالات.

إن إجراء **veb-TREE-PREDECESSOR** مناضّر لإجراء **veb-TREE-SUCCESSOR**، ولكن مع حالة إضافية.

**veb-TREE-PREDECESSOR**( $V, x$ )

```

1  if  $V.u == 2$ 
2      if  $x == 1$  and  $V.min == 0$ 
3          return 0
4      else return NIL
5  elseif  $V.max \neq \text{NIL}$  and  $x > V.max$ 
6      return  $V.max$ 
7  else  $min-low = \text{veb-TREE-MINIMUM}(V.cluster[high(x)])$ 
8      if  $min-low \neq \text{NIL}$  and  $low(x) > min-low$ 
9           $offset = \text{veb-TREE-PREDECESSOR}(V.cluster[high(x)], low(x))$ 
10         return index( $high(x), offset$ )
11     else  $pred-cluster = \text{veb-TREE-PREDECESSOR}(V.summary, high(x))$ 
12         if  $pred-cluster == \text{NIL}$ 
13             if  $V.min \neq \text{NIL}$  and  $x > V.min$ 
14                 return  $V.min$ 
15             else return NIL
16         else  $offset = \text{veb-TREE-MAXIMUM}(V.cluster[pred-cluster])$ 
17         return index( $pred-cluster, offset$ )

```

يكون السطران 13-14 الحالة الإضافية. تحدث هذه الحالة عندما لا يكون لاحق  $x$  - إن كان موجوداً - ضمن عنقود  $x$ . نؤكد لدينا في إجراء **veb-TREE-SUCCESSOR** أنه إذا كان لاحق  $x$  موجوداً خارج عنقود  $x$ ، فلا بد من أن يوجد في عنقود ذي رقم أعلى. لكن إذا كان سابق  $x$  هو القيمة الصغرى في شجرة  $V$  **veb**، فإن السابق لا يوجد في أيٍّ من العناقيد. يتفقد السطر 13 هذا الشرط، ويعيد السطر 14 القيمة الصغرى كما هو مطلوب.

لا تؤثر هذه الحالة الإضافية على زمن التنفيذ المقارب لزمن إجراء  $\text{vEB-TREE-PREDECESSOR}$ ، عند مقارنته بـ  $\text{vEB-TREE-SUCCESSOR}$ ، وهكذا يتقدّم  $\text{vEB-TREE-PREDECESSOR}$  بزمن  $O(\lg \lg u)$  في أسوأ الحالات.

### إدراج عنصر

نبحث الآن في كيفية إدراج عنصر ضمن شجرة  $\text{vEB}$ . نذكر هنا بأن  $\text{PROTO-VEB-INSERT}$  أجرى استدعاءين عوديين: أحدهما لإدراج العنصر والآخر لإدراج رقم عقود العنصر ضمن الملخص. سيقوم إجراء  $\text{vEB-TREE-INSERT}$  باستدعاء عودي واحد فقط. كيف نصل إلى استدعاء واحد فقط؟ عندما ندرج عنصرًا، فإما أن يوجد في العقود الذي يضاف إليه هذا العنصر، عنصر آخر سلفًا، وإما لا. فإذا تضمن العقود عنصرًا آخر سلفًا بالفعل، فإن رقم العقود موجود فعليًا في الملخص، وبذلك لا نحتاج إلى أن نحري ذلك الاستدعاء العودي. وإذا لم يتضمن العقود عنصرًا آخر سلفًا، فإن العنصر المُنزَج يصبح العنصر الوحيد في العقود، ولا نكون بحاجة للتكرار عوديًا لإدراج عنصر في شجرة  $\text{vEB}$  خالية:

$\text{vEB-EMPTY-TREE-INSERT}(V, x)$

- 1  $V.min = x$
- 2  $V.max = x$

إذا كان لدينا هذا الإجراء، نجد فيما يلي شبه الرماز لإجراء  $\text{vEB-TREE-INSERT}(V, x)$  الذي يفترض أن  $x$  ليس موجودًا بالفعل في المجموعة التي تمثلها شجرة  $\text{vEB } V$ :

$\text{vEB-TREE-INSERT}(V, x)$

- 1 if  $V.min == \text{NIL}$
- 2      $\text{vEB-EMPTY-TREE-INSERT}(V, x)$
- 3 else if  $x < V.min$
- 4     exchange  $x$  with  $V.min$
- 5     if  $V.u > 2$
- 6         if  $\text{vEB-TREE-MINIMUM}(V.cluster[\text{high}(x)]) == \text{NIL}$
- 7              $\text{vEB-TREE-INSERT}(V.summary, \text{high}(x))$
- 8              $\text{vEB-EMPTY-TREE-INSERT}(V.cluster[\text{high}(x)], \text{low}(x))$
- 9         else  $\text{vEB-TREE-INSERT}(V.cluster[\text{high}(x)], \text{low}(x))$
- 10     if  $x > V.max$
- 11          $V.max = x$

يعمل الإجراء كما يلي: يتحقق السطر 1 من أن  $V$  هي شجرة  $\text{vEB}$  خالية، فإذا كانت كذلك، يعالج السطر 2 هذه الحالة السهلة. تفترض السطور 3-11 أن  $V$  غير خالية، ولذلك فإن عنصرًا ما سيُنزَج في أحد عنايق  $V$ . ولكن قد لا يكون هذا العنصر هو العنصر  $x$  الذي مُرّر إلى  $\text{vEB-TREE-INSERT}$  بالضرورة. إذا

كان  $x < \min$  عند اختباره في السطر 3، فإن  $x$  يجب أن يُصبح  $\min$  الجديد. لكننا لا نريد أن نُضَيِّع  $\min$  الأصلي، ولهذا لا بد من أن ندرجه في أحد عناقيد  $V$ . في هذه الحالة، يبادل السطر 4 بين  $x$  و  $\min$  بحيث ندرج  $\min$  الأصلي في أحد عناقيد  $V$ .

لا نتخذ الأسطر 6-9 إلا إذا لم تكن  $V$  شجرة  $vEB$  في الحالة الأساسية. يحدد السطر 6: هل العقود الذي سيضاف فيه  $x$  خالي حاليًا؟ فإذا كان كذلك، يُدرج السطر 7 رقم عقود  $x$  في الملخص، ويالج السطر 8 الحالة البسيطة لإدراج  $x$  في عقود خالي. وإذا لم يكن عقود  $x$  خاليًا حاليًا، يُدرج السطر 9 العنصر  $x$  في عقوده. في هذه الحالة، لسنا بحاجة إلى تحديث الملخص، لأن رقم عقود  $x$  هو عنصر موجود فعليًا في الملخص.

أخيرًا، يتولى السطران 10-11 تحديث  $\max$  إذا كان  $x > \max$ . لاحظ أنه إذا كانت  $V$  هي شجرة  $vEB$  في الحالة الأساسية، أي ليست خالية، فإن الأسطر 3-4 و 10-11 تحدث  $\min$  و  $\max$  بصورة صحيحة.

مرة أخرى، يمكننا بسهولة أن نرى كيف توصف للمعادلة التكرارية (4.20) زمن التنفيذ. حسب نتيجة الاختبار في السطر 6، يجري تنفيذ الاستدعاء العودي في السطر 7 (التنفيذ على شجرة  $vEB$  مع حجم عالم يساوي  $\sqrt{u}$ )، أو الاستدعاء العودي في السطر 9 (التنفيذ على شجرة  $vEB$  مع حجم عالم يساوي  $\sqrt{u}$ ). في كلتا الحالتين، فإن الاستدعاء العودي الوحيد هو على شجرة  $vEB$  مع حجم عالم يساوي على الأكثر  $\sqrt{u}$ . ولما كان الباقي من  $vEB-TREE-INSERT$  يستغرق زمنًا  $O(1)$ ، فإننا نطبق للمعادلة التكرارية (4.20)، وبذلك يكون زمن التنفيذ  $O(\lg u)$ .

### حذف عنصر

أخيرًا، نلقي نظرة على كيفية حذف عنصر من شجرة  $vEB$ . يفترض الإجراء  $vEB-TREE-DELETE(V, x)$  أن  $x$  هو حاليًا عنصر موجود في المجموعة التي تمثلها شجرة  $vEB$ .

$vEB-TREE-DELETE(V, x)$

```

1  if  $V.min == V.max$ 
2       $V.min = NIL$ 
3       $V.max = NIL$ 
4  elseif  $V.u == 2$ 
5      if  $x == 0$ 
6           $V.min = 1$ 
7      else  $V.min = 0$ 
8           $V.max = V.min$ 
9  else if  $x == V.min$ 
10      $first-cluster = vEB-TREE-MINIMUM(V.summary)$ 
```

```

11  x = index(first-cluster,
12          vEB-TREE-MINIMUM(V.cluster[first-cluster]))
13  V.min = x
14  vEB-TREE-DELETE(V.cluster[high(x)], low(x))
15  if vEB-TREE-MINIMUM(V.cluster[high(x)]) == NIL
16      vEB-TREE-DELETE(V.summary, high(x))
17  if x == V.max
18      summary-max = vEB-TREE-MAXIMUM(V.summary)
19  if summary-max == NIL
20      V.max = V.min
21  else V.max = index(summary-max,
22          vEB-TREE-MAXIMUM(V.cluster[summary-max]))
23  elseif x == V.max
24      V.max = index(high(x),
25          vEB-TREE-MAXIMUM(V.cluster[high(x)]))

```

يعمل إجراء vEB-TREE-DELETE كما يلي: إذا تضمنت شجرة vEB  $V$  عنصراً واحداً فقط، فإن حذفه يتم بالسهولة التي جرى فيها إدراج عنصر في شجرة vEB خالية: إذ يكفي أن نجعل قيمة  $min$  و  $max$  مساوية لـ  $NIL$ . نعالج الأسطر 1-3 هذه الحالة. وفي الحالة المعاكسة، يكون في  $V$  عنصران على الأقل. يختار السطر 4 إذا كانت  $V$  شجرة vEB في الحالة الأساسية، فإذا كانت كذلك، نجعل الأسطر 5-8 قيمة  $min$  و  $max$  مساوية للعنصر الوحيد الباقي.

نفترض الأسطر 9-22 أن  $V$  تتضمن عنصرين أو أكثر وأن  $u \geq 4$ . علينا في هذه الحالة، أن نحذف عنصراً من العقود. ولكن قد لا يكون العنصر الذي نحذفه من العقود هو  $x$ ، لأنه في حال كان  $x$  يساوي  $min$  عندها، حالما نحذف  $x$ ، يصبح عنصراً آخر من أحد عنائيد  $V$  هو  $min$  الجديد، وعلينا أن نحذف هذا العنصر الآخر من عقوده. إذا أظهر الاختبار في السطر 9 أننا في هذه الحالة، عندها نضع السطر 10 في  $first-cluster$  رقم العقود الذي يتضمن العنصر الأصغري غير  $min$ ، ونضع السطر 11 في  $x$  قيمة العنصر الأصغري في ذلك العقود. يصبح هذا العنصر هو  $min$  الجديد في السطر 12، ولأننا وضعنا قيمته في  $x$ ، فسيكون هو العنصر الذي سيحذف من عقوده.

عندما نصل إلى السطر 13، نعلم أيضاً أنه يجب أن نحذف العنصر  $x$  من عقوده، سواء أكان  $x$  هو القيمة للمررة أصلاً إلى vEB-TREE-DELETE أم كان  $x$  هو العنصر الذي أصبح العنصر الأصغري الجديد. يحذف السطر 13 العنصر  $x$  من عقوده. ربما يكون هذا العقود قد أصبح خالياً، وهو ما يختاره السطر 14، فإذا كان كذلك، عندها نحتاج إلى حذف رقم عقود  $x$  من الملخص، وهو ما يعالجه السطر 15. بعد تحديث الملخص، ربما نحتاج إلى تحديث  $max$ . يتفقد السطر 16 حذف العنصر الأعظمي في  $V$ ، فإذا كان كذلك، نضع السطر 17 في  $summary-max$  رقم العقود غير الحالي ذي الرقم الأعلى. (يعمل

الاستدعاء  $VEB-TREE-MAXIMUM(V.summary)$  لأننا استدعينا  $VEB-TREE-DELETE$  عودياً على  $V.summary$ ، ومن ثم يكون  $V.summary.max$  قد حُدث سلفاً بالصورة للملائمة. إذا كانت جميع عناقد  $V$  خالية، عندما يكون العنصر الوحيد الباقي في  $V$  هو  $min$ ؛ يتحقق السطر 18 من هذه الحالة، ويُحدَّث السطر 19 قيمة  $max$  بالصورة للملائمة. في الحالة المعاكسة، يضع السطر 20 في  $max$  العنصر الأعظم من العنقود غير الخالي ذي الرقم الأعلى. (إذا كان العنصر قد حُدث من هذا العنقود، فنعمد ثانية على أن الاستدعاء العودي في السطر 13 قد صحح سلفاً واصفة  $max$  من ذلك العنقود.)

في النهاية، علينا أن نعالج الحالة التي لا يصبح فيها عنقود  $x$  خالياً نتيجة حذف  $x$ . رغم عدم ضرورة تحديثنا للملخص في هذه الحالة، إلا أنه قد يكون علينا تحديث  $max$ . يختار السطر 21 هذه الحالة، وإذا كان علينا تحديث  $max$ ، يقوم السطر 22 بذلك (مرة ثانية بالاعتماد على أن الاستدعاء العودي قد صحح  $max$  في العنقود).

نبين الآن أن  $VEB-TREE-DELETE$  يجري تنفيذه بزمن  $O(\lg \lg u)$  في أسوأ الحالات. قد يبدو للوهلة الأولى، أن المعادلة التكرارية (4.20) لا تُطبَّق دائماً لأن استدعاءً واحداً لـ  $VEB-TREE-DELETE$  قد ينشئ استدعاءين عوديين: أحدهما في السطر 13 والآخر في السطر 15. ومع أن الإجراء قد يُجري الاستدعاءين كليهما، فلنفكر في ما يحدث عندما يفعل ذلك. كي يحدث الاستدعاء العودي في السطر 15، يجب أن يبين الاختبار في السطر 14 أن عنقود  $x$  خالي. الحالة الوحيدة التي يمكن أن يكون فيها عنقود  $x$  خالياً هي إذا كان  $x$  هو العنصر الوحيد في عنقوده عندما قمنا بالاستدعاء العودي في السطر 13. لكن إذا كان  $x$  هو العنصر الوحيد في عنقوده، يكون ذلك الاستدعاء العودي قد تطلب زمناً  $O(1)$ ، لأنه نفذ الأسطر 1-3 فقط. بذلك، يكون لدينا احتمالان يستثني أحدهما الآخر:

- يستغرق الاستدعاء العودي في السطر 13 زمناً ثابتاً
- لم يحصل الاستدعاء العودي في السطر 15.

في كلتا الحالتين، تُوصَفُ المعادلةُ التكرارية (4.20) زمن تنفيذ  $VEB-TREE-DELETE$ ، وبذلك فإن زمن تنفيذه في أسوأ الحالات هو  $O(\lg \lg u)$ .

تمارين

1-3.20

عدّل أشجار  $VEB$  لتدعم المفاتيح المكررة.

2-3.20

عدّل أشجار  $VEB$  لتدعم المفاتيح التي لها معطيات تابعة مرتبطة.

## 3-3.20

اكتب شبه رماز لإجراء ينشئ شجرة van Emde Boas خالية.

## 4-3.20

ماذا يحدث لو استدعيت VEB-TREE-INSERT على عنصر موجود سابقاً في شجرة vEB؟ وماذا يحدث لو استدعيت VEB-TREE-DELETE على عنصر غير موجود في شجرة vEB؟ فسّر لماذا يسلك هذان الإجراءان هذا السلوك. بيّن كيف نعدّل أشجار vEB وعملياتها بحيث تتحقّق في زمن ثابت من وجود عنصر ما فيها.

## 5-3.20

افترض أننا أنشأنا، بدلاً من  $\sqrt{u}$  عقوداً حجم عالم كل منها  $\sqrt{u}$ ، أشجار vEB ليكون فيها  $u^{2/k}$  عقوداً، حجم عالم كل منها  $u^{1-1/k}$ ، حيث  $k$  ثابت أكبر من الواحد. إذا كان علينا أن نعدّل العمليات بصورة ملائمة، ماذا سيكون زمن تنفيذها؟ افترض بهدف التحليل، أن  $u^{2/k}$  و  $u^{1-1/k}$  أعداد صحيحة دوماً.

## 6-3.20

إن إنشاء شجرة vEB حجم عالمها  $u$  يتطلب زمناً  $\Theta(u)$ . افترض أننا نرغب بحساب تفصيلي لهذا الزمن. ما هو أصغر عدد عمليات  $n$  بحيث تستغرق كل عملية في شجرة vEB ما زمناً محبّذاً  $O(\lg \lg u)$ ؟

## مسائل

## 1-20 متطلبات الحجم لأشجار van Emde Boas

تسير هذه المسألة متطلبات الحجم لأشجار van Emde Boas، وتقدّم طريقة لتعديل بنية المعطيات لجعل متطلبات الحجم يعتمد على عدد العناصر  $n$  المخزنة حالياً في الشجرة، وليس على حجم العالم  $u$ . نفترض للتبسيط أن  $\sqrt{u}$  دائماً عدد صحيح.

أ. فسّر لماذا توصّف المعادلة التكرارية التالية للمتطلب الحجمي  $P(u)$  لشجرة van Emde Boas حجم عالمها  $u$ :

$$P(u) = (\sqrt{u} + 1)P(\sqrt{u}) + \Theta(\sqrt{u}). \quad (5.20)$$

ب. أثبت أن للمعادلة التكرارية (5.20) الحل  $P(u) = O(u)$ .

لتقليص المتطلبات الحجمية، نعرّف شجرة ذات حجم مقلّص *reduced-space van Emde Boas*، أو شجرة *RS-VEB*، بأنها شجرة vEB  $V$  مع التغيرات التالية:

- الواسفة  $V.cluster$  بدلاً من أن تكون عذنة كصيغة بسيطة من المؤشرات على أشجار  $vEB$  مع حجم عالم  $\sqrt{u}$ ، هي الآن جدول التلييد  $hash\ table$  (انظر الفصل 11) مخزن كجدول ديناميكي (انظر المقطع 4.17). مخزن جدول التلييد، كما في نسخة الصيغة من  $V.cluster$ ، مؤشرات على أشجار  $RS-vEB$  بحجم عالم  $\sqrt{u}$ . لإيجاد العنقود ذي الترتيب  $i$ ، نبحث عن المفتاح  $i$  في جدول التلييد، وبذلك يمكننا إيجاد العنقود ذي الترتيب  $i$  يبحث واحد في جدول التلييد.
  - مخزن جدول التلييد مؤشرات إلى العناقيد غير الحالية فقط. بعيد البحث عن عنقود خالي في جدول التلييد القيمة  $NIL$ ، مبيّن أن العنقود خال.
  - تأخذ الواسفة  $V.summary$  القيمة  $NIL$  إذا كانت جميع العناقيد الحالية. في الحالة للمعكسة، يشير  $V.summary$  إلى شجرة  $RS-vEB$  بحجم عالم  $\sqrt{u}$ .
- لما كان جدول التلييد يُنخَر باستخدام جدول ديناميكي، فإن الحجم الذي يتطلبه متناسب طرذاً مع عدد العناقيد غير الحالية.
- عندما نحتاج إلى إدراج عنصر في شجرة  $RS-vEB$  خالية، ننشئ شجرة  $RS-vEB$  باستدعاء الإجراء التالي، حيث المتوسط  $u$  هو حجم العالم لشجرة  $RS-vEB$ .

#### CREATE-NEW-RS-VEB-TREE( $u$ )

```

1  allocate ■ new vEB tree V
2  V.u = u
3  V.min = NIL
4  V.max = NIL
5  V.summary = NIL
6  create V.cluster as an empty dynamic hash table
7  return V

```

ت. عدّل إجراء  $vEB-TREE-INSERT$  لتوليد شبه رماز لإجراء  $RS-vEB-TREE-INSERT(V, x)$  الذي يدرج  $x$  في شجرة  $V$  من نمط  $RS-vEB$ ، مستدعيًا  $CREATE-NEW-RS-VEB-TREE$  عند اللزوم.

ث. عدّل إجراء  $vEB-TREE-SUCCESSOR$  لتوليد شبه رماز لإجراء  $RS-vEB-TREE-SUCCESSOR(V, x)$  الذي يعيد العنصر اللاحق لـ  $x$  في شجرة  $V$  من نمط  $RS-vEB$ ، أو  $NIL$  إذا لم يكن لـ  $x$  عنصر لاحق في  $V$ .

ج. أثبت - بافتراض أن التلييد بسيط ومنظم - أن إجراءي  $RS-vEB-TREE-INSERT$  و  $RS-vEB-TREE-SUCCESSOR$  ينفذان بزمن متوقع مخفّد  $O(\lg \lg u)$ .

ح. بافتراض أن العناصر لا تُحذف أبداً من شجرة  $vEB$ ، أثبت أن لتطلب الحجمي لبنية شجرة  $RS-vEB$  هو  $O(n)$ ، حيث  $n$  هو عدد العناصر المخزنة حالياً في شجرة  $RS-vEB$ .



خ. الأشجار RS-VEB لها ميزة أخرى على أشجار vEB: حيث يتطلب إنشاؤها زمنًا أقل. كم يستغرق إنشاء شجرة RS-VEB فارغة؟

## 2-20 بنية y-fast tries

تبحث هذه المسألة في بنية "y-fast tries" التي اقترحها D. Willard، والتي تُنفَّذ، مثل أشجار van Emde Boas، كلاً من العمليات MEMBER و MINIMUM و MAXIMUM و PREDESSOR و SUCCESSOR على عناصر مأخوذة من عالم حجمه  $u$  بزمن  $O(\lg \lg u)$  في أسوأ الحالات. تستغرق عمليتا INSERT و DELETE زمنًا محضًا  $O(\lg \lg u)$ . تُستخدم y-fast tries، حجمًا  $O(n)$  فقط لتخزين  $n$  عنصرًا مثل أشجار reduced-space van Emde Boas (انظر المسألة 1-20). يعتمد تصميم y-fast tries على التلييد المثالي perfect hash (انظر المقطع 5.11).

في بنية مبدئية، افترض أننا ننشئ جدول تلييد مثالي لا يتضمن جميع العناصر في المجموعة الديناميكية فقط، بل يتضمن كلًا سابقة prefix من التمثيل التالي لكل عنصر في المجموعة. فمثلاً، إذا كان  $u = 16$ ، فإن  $\lg u = 4$ ، والعنصر  $x = 13$  موجود في المجموعة. ولما كان التمثيل الثنائي لـ 13 هو 1101، فإن جدول التلييد المثالي سيتضمن المتواليات 1 و 11 و 110 و 1101. ننشئ إضافةً إلى جدول التلييد قائمة مضاعفة الترابط من العناصر الموجودة حاليًا في المجموعة، بترتيب متزايد.

أ. ما الحجم الذي تتطلبه هذه البنية؟

ب. بَيِّن كيفية إنجاز عمليتي MINIMUM و MAXIMUM بزمن  $O(1)$  وعمليات MEMBER و PREDECESSOR و SUCCESSOR بزمن  $O(\lg \lg u)$  وعمليات INSERT و DELETE بزمن  $O(\lg u)$ .

لتقليل المتطلب الحجمي إلى  $O(n)$ ، نقوم بالتعديلات التالية على بنية المعطيات:

- نُنفِّذُ العناصر التي عددها  $n$  ضمن  $n/\lg u$  مجموعة ذات حجم  $\lg u$ . (افترض الآن أن  $\lg u$  تُقسِمُ  $n$ ). تتألف المجموعة الأولى من  $\lg u$  أصغر عنصر في المجموعة، وتتألف المجموعة الثانية من  $\lg u$  أصغر عنصر مما تبقى، وهكذا.
- نعيِّن قيمةً "مثلة" عن كل مجموعة. تكون قيمة مثل المجموعة ذات الترتيب  $i$  مساوية لقيمة أكبر عنصر في المجموعة  $i$  على الأقل، وهي كذلك أصغر من جميع عناصر المجموعة  $(i + 1)$ . (يمكن أن يكون مثل آخر مجموعة أكبر عنصر ممكن  $u$  مفقودًا منه 1.) لاحظ أن المثل يمكن أن يكون قيمة غير موجودة حاليًا في المجموعة.
- نُخزِّنُ الـ  $\lg u$  عنصرًا من كل مجموعة في شجرة بحث ثنائية متوازنة، مثل شجرة حمراء-سوداء. يشير كل مثل إلى شجرة البحث الثنائية المتوازنة الخاصة بمجموعتها، وتُشير كل شجرة بحث ثنائية متوازنة إلى مثل مجموعتها.

■ يُخزّن جدولُ التبيد للثالي المُعلّين فقط، كما يخزنون أيضًا في قائمة مضاعفة الارتباط بترتيب متزايد.

نسمي هذه البنية *y-fast trie*.

ت. بَيِّنْ أن *y-fast trie* تتطلب حجمًا  $O(n)$  فقط لتخزين  $n$  عنصرًا.

ث. بَيِّنْ كيفية إنجاز عمليتي MINIMUM و MAXIMUM بزمن  $O(\lg \lg u)$  باستخدام *y-fast trie*.

ج. بَيِّنْ كيفية إنجاز عملية MEMBER بزمن  $O(\lg \lg u)$ .

ح. بَيِّنْ كيفية إنجاز عمليتي PREDECESSOR و SUCCESSOR بزمن  $O(\lg \lg u)$ .

خ. اشرح لماذا تستغرق عمليتا INSERT و DELETE زمنًا  $\Omega(\lg \lg u)$ .

د. بَيِّنْ كيفية إرخاء relax مطلب أن يكون عدد عناصر كل مجموعة من *y-fast-trie* هو  $\lg u$  عنصرًا تمامًا ليسمح بتنفيذ INSERT و DELETE بزمن مخشد  $O(\lg \lg u)$  دون التأثير في الأمانة المقاربة لتنفيذ العمليات الأخرى.

## ملاحظات الفصل

سُمِّيت بنية المعطيات في هذا الفصل باسم P. van Emde Boas، الذي وَصَفَ صيغةً أوليةً للفكرة في عام 1975 [339]. فصلت المقالات اللاحقة لـ van Emde Boas [340] و van Emde Boas و Kass و Zijlstra [341] الفكرة والعرض. وسَّع Näher و Mehlhorn [252] لاحقًا هذه الأفكار لتطبيقات على حجوز عالم أولية. يتضمن كتاب Mehlhorn [249] معالجة لأشجار van Emde Boas تختلف قليلًا عما هو موجود في هذا الفصل.

قام Dementiev وزملاؤه [84] باستخدام الأفكار المتعلقة بأشجار van Emde Boas لتطوير شجرة بحث من ثلاثة مستويات غير عودية تُنفَّذ بصورة أسرع من أشجار van Emde Boas في تجاربهم الخاصة.

صمَّم wang and Lin [347] إصدارًا عتاديًا أنبويًا hardware-pipelined من أشجار van Emde Boas، يحقق زمنًا مخشدًا ثابتًا لكل عملية وتستخدم  $O(\lg \lg u)$  مرحلة في الأنابيب pipeline.

بيَّرن حدَّ أدنى اكتشفه Thorup و Pătrașcu [273, 274] للشور على العنصر السابق predecessor، أن أشجار van Emde Boas مثلى لهذه العملية، ولو كانت العشوائية مسموحة.

## 21 بنى المعطيات للمجموعات المنفصلة

تطلب بعض التطبيقات تجميع  $n$  عنصرًا متمايزًا في تجميع من المجموعات المنفصلة. تحتاج هذه التطبيقات غالبًا إلى إجراء عمليتين هما: إيجاد المجموعة الوحيدة التي ينتمي إليها عنصر ما، وتوحيد مجموعتين. يستكشف هذا الفصل طرق الحفاظ على بنية معطيات تدعم هذه العمليات.

يصف المقطع 1.21 العمليات التي تدعمها بنية معطيات مجموعات منفصلة، ويقدم تطبيقًا بسيطًا. وفي المقطع 2.21 ندرس تنجيّرًا بسيطًا للمجموعات المنفصلة باستخدام اللائحة المترابطة. يعطي المقطع 3.21 تمثيلًا أكثر فعالية باستخدام الأشجار ذات الجذور. إنّ زمن التنفيذ باستخدام التمثيل الشجري هو نظريًا فوق خطي، لكنه خطي في جميع الأهداف العملية. يعرف المقطع 4.21 دالة سريعة النمو، ودالتها للمعكسة البطيئة النمو التي تظهر في زمن تنفيذ العمليات على التنجيز الشجري وتناقشهما، ثم يُثبت - بالتحليل المحدث - حدًا أعلى لزمن التنفيذ الذي هو بالكاد فوق خطي.

### 1.21 عمليات المجموعات المنفصلة

تحتوي بنية معطيات المجموعات المنفصلة *disjoint-set data structure* على تجميع  $S = \{S_1, S_2, \dots, S_R\}$  من المجموعات الديناميكية المنفصلة. تُعدّ كل مجموعة بممثل *representative* هو أحد عناصرها. في بعض التطبيقات، لا يُعدّ العنصر الذي استخدم تمثيلًا أمرًا مهمًا؛ بل المهم أننا إذا طلبنا ممثل مجموعة ديناميكية مرتين دون تعديل المجموعة بينهما هو أن نحصل على الإجابة نفسها. قد تتطلب تطبيقات أخرى وجود قاعدة محدّدة سلفًا لاختيار الممثل، كاختيار العنصر الأصغر في المجموعة (طبقًا بافتراض إمكان ترتيب العناصر).

وكما في تنجيزات المجموعات الديناميكية الأخرى التي درستناها، تُمثل كل عنصر في المجموعة بفرض object. فإذا كان  $x$  غرضًا، فإننا نرغب بدعم العمليات التالية:

(MAKE-SET( $x$ ) إنشاء مجموعة جديدة فيها عنصر وحيد هو  $x$  (ومن ثم فهو الممثل). ولما كانت المجموعات منفصلة، فإننا نطلب ألا يكون  $x$  موجودًا سلفًا في مجموعة أخرى.

$UNION(x, y)$  الاجتماع الذي يجمع المجموعتين الديناميكيتين اللتين تحتويان على  $x$  و  $y$  ( $S_x$  و  $S_y$  مثلاً) في مجموعة جديدة هي اجتماعهما. نفترض أن المجموعتين منفصلتان قبل العملية. إن تمثل المجموعة الناتجة هو أي عنصر من  $S_y \cup S_x$ ، علمًا بأن العديد من التجزئات يختار ممثل  $S_x$  أو  $S_y$  ليكون ممثلاً للمجموعة الجديدة. ولما كان المطلوب هو أن تكون المجموعات في التجمع منفصلة، فإننا ندمر destroy المجموعتين  $S_x$  و  $S_y$  مفاهيميًا ونخذفهما من التجمع. وغالبًا ما نقوم عمليًا بامتصاص absorb عناصر إحدى المجموعتين ضمن المجموعة الأخرى.

$FIND-SET(x)$  إيجاد مجموعة تعيد مؤشرًا إلى مثل المجموعة (الوحيدة) التي تحتوي على  $x$ .

سنحلل في هذا الفصل أزمنة تنفيذ العمليات على بنية معطيات المجموعات المنفصلة بدلالة وسطيتهما:  $n$  عدد عمليات MAKE-SET و  $m$  العدد الكلي للعمليات MAKE-SET و UNION و FIND-SET. ولما كانت المجموعات منفصلة، فإن كل عملية UNION تقلص عدد المجموعات بمقدار واحد. ولذلك، وبعد  $n-1$  عملية تبقى لدينا مجموعة واحدة فقط. ومن ثم، فإن عدد عمليات UNION هو على الأكثر  $n-1$ . لاحظ أيضًا أنه لما كانت عمليات MAKE-SET متضمنة في العدد الكلي للعمليات  $m$ ، فإن  $m \geq n$ . نفترض أن عمليات MAKE-SET هي العمليات  $n$  التي للنجزة أولاً.

### تطبيق على بنى معطيات المجموعات المنفصلة

يظهر أحد التطبيقات المتعددة لبنى معطيات المجموعات المنفصلة في تحديد المكونات المرتبطة في بيان غير موجه (انظر المقطع ب.4). فمثلاً، يظهر الشكل 1.21 أ) بياناً مؤلفاً من أربعة مكونات. يستخدم الإجراء CONNECTED-COMPONENTS التالي عمليات المجموعات المنفصلة لحساب المكونات المرتبطة في بيان. بمجرد تنفيذ CONNECTED-COMPONENTS لمعالجة أولية للبيان، يقوم الإجراء SAME-COMPONENT بالإجابة عن الاستعلامات المتعلقة بكون عقدتين تنتميان إلى المكون نفسه.<sup>1</sup> (نرمز في شبه الرماز إلى مجموعة عقد بيان  $G$  بـ  $G.V$ ، وإلى مجموعة الوصلات بـ  $G.E$ ).

#### CONNECTED-COMPONENTS( $G$ )

- 1 for each vertex  $v \in G.V$
- 2 MAKE-SET( $v$ )
- 3 for each edge  $(u, v) \in G.E$

<sup>1</sup> عندما تكون وصلات البيان سكونية (أي لا تتغير مع الزمن)، يمكننا حساب المكونات المرتبطة بسرعة أكبر باستخدام البحث عمقاً-أولاً (التسعين 12-3.22). مع ذلك، تضاف الوصلات ديناميكياً أحياناً ونحتاج إلى الحفاظ على المكونات المرتبطة مع إضافة كل وصلة. في هذه الحالة، يمكن أن يكون التجزير الموجود هنا أكثر فعالية من تنفيذ بحث جديد عمقاً-أولاً لكل وصلة جديدة.

```

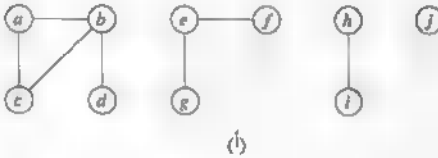
4   if FIND-SET(u) ≠ FIND-SET(v)
5       UNION(u, v)
    
```

SAME-COMPONENT(u, v)

```

1   if FIND-SET(u) == FIND-SET(v)
2       return TRUE
3   else return FALSE
    
```

يضع الإجراء CONNECTED-COMPONENTS في البداية كل عقدة  $v$  في مجموعة خاصة بها. ثم يوحد unite المجموعتين اللتين تحتويان  $u$  و  $v$ ، وذلك لكل وصلة  $(u, v)$ . يُطلب في التمرين 1.21-2، بعد معالجة جميع الوصلات، إثبات أن عقدتين تقعان في المكون نفسه إذا وفقط إذا كان الفرضان المقابلان لهما في المجموعة نفسها. ومن ثم، يُحسب الإجراء CONNECTED-COMPONENTS المجموعات بطريقة تجعل الإجراء SAME-COMPONENT يحدّد وجود عقدتين في المكون المرتبط نفسه. يبين الشكل 1.21(ب) كيف يُحسب CONNECTED-COMPONENTS المجموعات المنفصلة.



تجميع المجموعات المنفصلة										الوصلة للمعالجة
{j}	{i}	{h}	{g}	{f}	{e}	{d}	{c}	{b}	{a}	المجموعات الابتدائية
{j}	{i}	{h}	{g}	{f}	{e}		{c}	{b, d}	{a}	{b, d}
{j}	{i}	{h}		{f}	{e, g}		{c}	{b, d}	{a}	{e, g}
{j}	{i}	{h}		{f}	{e, g}			{b, d}	{a, c}	{e, c}
{j}		{h, i}		{f}	{e, g}			{b, d}	{a, c}	{h, i}
{j}		{h, i}		{f}	{e, g}				{a, b, c, d}	{a, b}
{j}		{h, i}			{e, f, g}				{a, b, c, d}	{e, f}
{j}		{h, i}			{e, f, g}				{a, b, c, d}	{b, c}

(ب)

الشكل 1.21 (أ) يبين مؤلف من أربعة مكونات  $\{a, b, c, d\}$  و  $\{e, f, g\}$  و  $\{h, i\}$  و  $\{j\}$ . (ب) تجميع المجموعات المنفصلة بعد معالجة كل وصلة.

في تنجيز فعلي لخوارزمية المكونات المرتبطة هذه، سيحتاج كلٌّ من تمثيلي البيان وبنية معطيات المجموعات المنفصلة إلى أن يشير كلٌّ منهما إلى الآخر. أي إنَّ كلَّ غرضي يمثِّل عقدةً سيحتوي على مؤشر *pointer* إلى غرضي المجموعة المنفصلة للقابل، والعكس بالعكس. تعتمد هذه التفاصيل البرمجية على لغة التنجيز، ولن نعالجها أكثر من ذلك هنا.

تمارين

1-1.21

افترض أنَّه جرى تنفيذ **CONNECTED-COMPONENTS** على البيان غير الموجه  $G = (V, E)$ ، حيث  $V = \{a, b, c, d, e, f, g, h, i, j, k\}$  ونجري معالجة الوصلات  $E$  بالترتيب التالي:  $(d, i), (f, k), (g, i), (b, g), (a, h), (i, j), (d, k), (b, j), (d, f), (g, j), (a, e)$  اسرد العقد في كلِّ مكونٍ مرتبط بعد كل تكرار للأسطر 5-3.

2-1.21

بيِّن أنه بعد معالجة جميع الوصلات في **CONNECTED-COMPONENTS**، تكون عقدتان في المكون نفسه إذا وفقط إذا كانا في المجموعة نفسها.

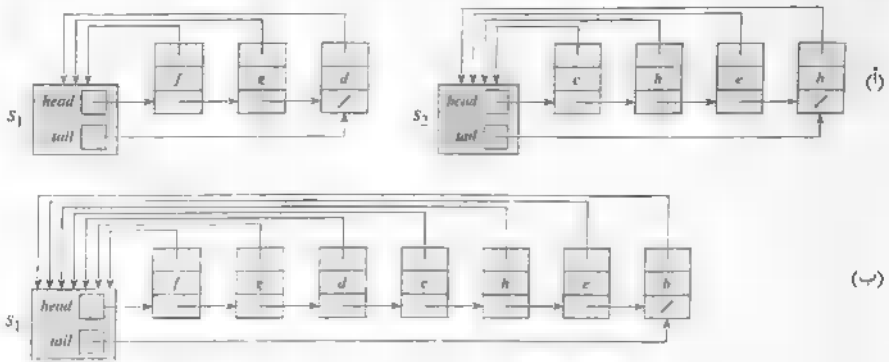
3-1.21

ما هو عدد المرات التي يُستدعى فيها **FIND-SET** خلال تنفيذ **CONNECTED-COMPONENTS** على بيان غير موجه  $G = (V, E)$  ذي  $k$  مكونًا مرتبطًا؟ وما هو عدد المرات التي يُستدعى فيها **UNION**؟ عبِّر عن أجوبتك بدلالة  $|V|$  و  $|E|$  و  $k$ .

## 2.21 تمثيل المجموعات المنفصلة باللائحة مترابطة

يُظهر الشكل 2.21(أ) طريقةً بسيطةً لتنجيز بنية معطيات مجموعات منفصلة: تُثَّل فيها كلُّ مجموعة باللائحة المترابطة الخاصة بها. يحتوي الغرض في كلِّ مجموعة على واصفات رأس *head* يُوْشِر على الغرض الأول في اللائحة، وذيل *tail* يُوْشِر على الغرض الأخير. يحتوي كلُّ غرضي من اللائحة المترابطة على عنصر من المجموعة، ومؤشِر إلى الغرض التالي في اللائحة، ومؤشِر يرجع إلى غرض المجموعة. يمكن أن تظهر الأغراض في كل لائحة مترابطة بأي ترتيب كان. إنَّ تمثِّل المجموعة هو عنصرُ المجموعة في الغرض الأول في اللائحة.

من السهل تنفيذ **MAKE-SET** و **FIND-SET** كليهما بزمن  $O(1)$  في تمثيل اللائحة المترابطة. ولتنفيذ **MAKE-SET(x)** ننشئ لائحة مترابطة جديدة غرضها الوحيد هو  $x$ . ففي حالة **FIND-SET(x)** نعيد المؤشِر من  $x$  إلى غرض المجموعة الخاص بها، ثم نعيد العنصر في الغرض الذي يشير إليه *head*. فمثلاً، في الشكل 2.21(ب)، يعيد استدعاء **FIND-SET(g)** القيمة  $f$ .



(ب)

**الشكل 2.21 (أ)** تمثيل مجموعتين باستخدام اللائحة المترابطة. تحتوي المجموعة  $S_1$  على العناصر  $d$  و  $e$  و  $f$  و  $g$  حيث  $f$  هو الممثل. وتحتوي المجموعة  $S_2$  على العناصر  $b$  و  $c$  و  $h$  حيث  $h$  هو الممثل. يحتوي كل غرض في اللائحة على عنصر من المجموعة، ومؤشر إلى الغرض التالي في اللائحة، ومؤشر يرجع إلى غرض المجموعة. ويحتوي كل غرض مجموعة على مؤشر  $head$  ومؤشر  $tail$  على الغرض الأول والغرض الأخير على التوالي. (ب) نتيجة  $UNION(g, e)$  الذي يُلحق اللائحة المترابطة التي تحتوي  $e$  باللائحة المترابطة التي تحتوي  $g$ . تمثل اللائحة الناتجة هو  $f$ . ويجري تدمير  $S_2$ ، غرض المجموعة الخاصة باللائحة  $e$ .

### تدوير بسيط للإجراء UNION

يستغرق التدوير الأبسط لعملية UNION باستخدام تمثيل المجموعة باللائحة مترابطة وقتاً أكثر بكثير من  $FIND-SET$  و  $MAKE-SET$ . وكما يظهر الشكل 2.21 (ب) فإننا ننفذ  $UNION(x, y)$  بإلحاق لائحة  $y$  بنهاية لائحة  $x$ . ويصبح ممثل لائحة  $x$  هو الممثل للمجموعة الناتجة. نستخدم المؤشر  $tail$  في لائحة  $x$  لإيجاد مكان إلحاق باللائحة  $y$  بسرعة. ولما كانت جميع عناصر لائحة  $y$  ستندمج إلى لائحة  $x$ ، فيمكننا تدمير غرض المجموعة للائحة  $y$ . ولكن علينا تحديث المؤشر إلى غرض المجموعة لكل غرض كان في الأصل في لائحة  $y$ ، وهذا يستغرق زمناً خطئياً نسبة إلى طول اللائحة  $y$ . ففي الشكل 2.21 مثلاً، تسبب عملية  $UNION(g, e)$  تعديل المؤشرات في الأغراض  $b$  و  $c$  و  $e$  و  $h$ .

في الحقيقة، ليس من السهل إنشاء متتالية من  $m$  عملية على  $n$  غرضاً تتطلب زمناً  $\Theta(n^2)$ . افترض أن لدينا الأغراض  $x_1, x_2, \dots, x_n$ . ننفذ متتالية من  $n$  عملية  $MAKE-SET$  تتبعها  $n-1$  عملية  $UNION$  المبينة في الشكل 3.21، بحيث يكون  $m = 2n - 1$ . يستغرق إجراء  $n$  عملية  $MAKE-SET$  زمناً رتبته  $\Theta(n)$ . ولما كانت عملية  $UNION$  ذات الرقم  $i$  تحدث  $i$  غرضاً، فإن العدد الكلي للأغراض المحدثة في  $n-1$  عملية  $UNION$  هو:

$$\sum_{i=1}^{n-1} i = \Theta(n^2).$$

العملية	عدد الأغراض المحددة
MAKE-SET( $x_1$ )	1
MAKE-SET( $x_2$ )	1
$\vdots$	
MAKE-SET( $x_n$ )	1
UNION( $x_2, x_3$ )	1
UNION( $x_3, x_2$ )	2
UNION( $x_4, x_3$ )	3
$\vdots$	
UNION( $x_n, x_{n-1}$ )	$n - 1$

**الشكل 3.21** متتالية من  $2n - 1$  عملية على  $n$  غرضًا تستغرق زمنًا  $\Theta(n^2)$ ، أو زمنًا  $\Theta(n)$  لكل عملية وسطيًا، باستخدام تمثيل المجموعات باللائحة مترابطة والتتبع البسيط لـ UNION.

ويكون العدد الكلي للعمليات هو  $2n - 1$  عملية، وبذلك تستغرق كلُّ عملية زمنًا وسطيًا  $\Theta(n)$ . أي إنَّ الزمن المعتد لعملية ما هو  $\Theta(n)$ .

### كسبية اجتماع مثقل

يتطلب التتبع السابق للإجراء UNION في أسوأ الحالات زمنًا  $\Theta(n)$  لكل استدعاء وسطيًا، لأننا قد نلحق لائحة طويلة بلائحة قصيرة؛ يجب أن نحدِّث المؤشر إلى المثل، لكلِّ عنصرٍ في اللائحة الطويلة. افترض بدلًا من ذلك، أنَّ كلَّ لائحة تتضمن أيضًا طول اللائحة (الذي يمكن الحفاظ عليه بسهولة) وأننا نلحق اللائحة الأقصر باللائحة الأطول دومًا مع كسر الروابط اعتباطيًا. بكسبية الاجتماع المثقل *weighted-union heuristic* البسيطة هذه يمكن أن تقلل عملية UNION بحاجة إلى زمن  $\Omega(n)$  إذا كانت المجموعتان تحتويان  $\Omega(n)$  عنصرًا. ومع ذلك، وكما تبينَّ المبرهنة التالية، فإنَّ متتالية مؤلفة من  $m$  عملية MAKE-SET و UNION و FIND-SET منها  $n$  عملية MAKE-SET تستغرق زمنًا  $O(m + n \lg n)$ .

### مبرهنة 1.21

باستخدام تمثيل اللائحة المترابطة للمجموعات المنفصلة وكسبية الاجتماع المثقل، تستغرق متتالية مؤلفة من  $m$  عملية MAKE-SET و UNION و FIND-SET منها  $n$  عملية MAKE-SET زمنًا  $O(m + n \lg n)$ .

**البرهان** لما كانت عملية UNION تجمع مجموعتين منفصلتين، فإننا نُنجز  $n - 1$  عملية UNION على الأكثر. نقوم الآن بحد الزمن الكلي الذي تستغرقه عمليات UNION هذه. نبدأ بتحديد حدٍّ أعلى لعدد المرات التي يجري فيها تحديث مؤشرات الأغراض إلى أغراض المجموعات الخاصة بها. ليكن لدينا غرض عدد  $x$ . نعلم أنَّه في كلِّ مرة يُحدِّث فيها مؤشر  $x$ ، يجب أن يكون  $x$  قد بدأ في المجموعة الصغرى. في المرة الأولى



التي جرى فيها تحديث مؤشر  $x$ ، يجب أن يكون في المجموعة الناتجة عنصران على الأقل. وبالمثل عند تحديث مؤشر  $x$  في المرة التالية، يجب أن يكون في المجموعة الناتجة أربعة عناصر على الأقل. وبالتابعة على هذا المنوال، نلاحظ أنه بعد تحديث مؤشر  $x$   $\lg k$  مرة، يجب أن يكون في المجموعة الناتجة  $k$  عنصرًا على الأقل، لأي  $k \leq n$ . ولما كانت المجموعة الكبرى تحوي  $n$  عنصرًا على الأكثر، فإن كل مؤشر إلى غرض يكون قد جرى تحديثه  $\lg n$  مرة على الأكثر في كل عمليات UNION. وهكذا يكون الزمن الكلي المصروف لتحديث مؤشرات الأغراض في كل عمليات UNION هو  $O(n \lg n)$ . يجب أن نحسب أيضًا تحديث المؤشرات *tail* وأطوال اللاتحة التي تستغرق زمنًا  $\Theta(1)$  لكل عملية UNION. وبذلك يكون الزمن الكلي المصروف في تحديث  $n$  غرضًا هو  $O(n \lg n)$ .

يمكن استنتاج الزمن الكامل لمتتالية العمليات  $m$  بسهولة؛ فكل عملية MAKE-SET و FIND-SET تستغرق زمنًا  $O(1)$ ، وعدد العمليات هو  $O(m)$ . إذن الزمن الكلي للمتتالية كاملة هو  $O(m + n \lg n)$ . ■

### تعاريف

#### 1-2.21

اكتب شبه رماز لكل من MAKE-SET و FIND-SET و UNION باستخدام تمثيل اللاتحة المترابطة وكسبية الاجتماع المنثقل. تأكد أنك حددت الواصفات التي افترضتها لأغراض المجموعات وأغراض اللوائح.

#### 2-2.21

بين بنية المعطيات الناتجة عن عمليات FIND-SET والإجابات المعادة في البرنامج التالي. استخدم تمثيل اللاتحة المترابطة مع كسبية الاجتماع المنثقل.

```

1  for  $l = 1$  to 16
2      MAKE-SET( $x_l$ )
3  for  $l = 1$  to 15 by 2
4      UNION( $x_l, x_{l+1}$ )
5  for  $l = 1$  to 13 by 4
6      UNION( $x_l, x_{l+2}$ )
7  UNION( $x_1, x_5$ )
8  UNION( $x_{11}, x_{13}$ )
9  UNION( $x_1, x_{10}$ )
10 FIND-SET( $x_2$ )
11 FIND-SET( $x_9$ )

```

افترض أنه إذا كان للمجموعتين  $x_i$  و  $x_j$  الحجم نفسه، فإن العملية  $\text{UNION}(x_i, x_j)$  تُلحق لاتحة  $x_j$  بالاتحة  $x_i$ .

#### 3-2.21

عدّل البرهان التجميعي للمبرهنة 1.21 للحصول على الحدود الزمنية للمخّذة  $O(1)$  لكل من MAKE-SET

و FIND-SET والحد  $O(\lg n)$  UNION باستخدام تمثيل اللائحة المترابطة وكسبية الاجتماع المثقل.

#### 4-2.21

أعطِ حلاً مقارناً مُحْكَمًا لزمَن تنفيذ متتالية العمليات في الشكل 3.21 بافتراض تمثيل اللائحة المترابطة وكسبية الاجتماع المثقل.

#### 5-2.21

يظن الأستاذ Gompers أنه قد يكون من الممكن الاحتفاظ بمؤشر واحد فقط في كل غرض مجموعة، بدلاً من اثنين (head و tail)، مع الاحتفاظ بمؤشرين في كل عنصر من عناصر اللائحة. يرى أن ظن الأستاذ مبيّئ على أسس جيدة، عن طريق وصف كيفية تمثيل مجموعة باللائحة مترابطة بحيث يكون للعمليات زمن تنفيذ العمليات الموصوفة في هذا المقطع نفسه. صِف أيضاً كيفية عمل العمليات. يجب أن يسمح أسلوبك باستخدام كسبية الاجتماع المثقل، بالتأثير للموصف في هذا المقطع نفسه. (تلميح: استنخدم ذيل اللائحة المترابطة باعتباره مثالاً لمجموعتها.)

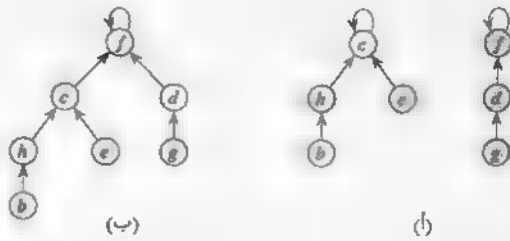
#### 6-2.21

اقترح تغييراً بسيطاً للإجراء UNION في تمثيل اللائحة المترابطة يلغي الحاجة إلى الاحتفاظ بالمؤشر tail إلى الغرض الأخير في اللائحة. يجب ألا يتغير زمن التنفيذ للمقارب للإجراء UNION سواء استُخدمت كسبية الاجتماع المثقل أم لم تُستخدم. (تلميح: بدلاً من إلقاء لائحة بأخرى، صلّها معاً.)

### 3.21 غابات المجموعات المنفصلة

في تنجيز أسرع للمجموعات المنفصلة، نحل المجموعات بأشجار ذات جذور بحيث تحتوي كل عقدة على عنصر، ونُقل كل شجرة مجموعة. في غابة المجموعات المنفصلة disjoint-set forest الموضحة في الشكل 4.21(أ)، يشير كل عنصر إلى أبيه فقط. يتضمن جذر كل شجرة تمثل المجموعة وهو أب لنفسه. وكما سنرى لاحقاً، ومع أن الخوارزميات المباشرة التي تُستخدم هذا التمثيل ليست أسرع من تلك التي تُستخدم تمثيل اللائحة المترابطة، فيمكننا بإدخال كسبيتين (هما الاجتماع بحسب المرتبة، وضغط المسار) الوصول إلى بنية معطيات أمثلية للمجموعات المنفصلة (بالمقارنة).

ننفذ العمليات الثلاث على المجموعات المنفصلة كما يلي. نُنشئ عملية MAKE-SET شجرة ذات عقدة واحدة فقط ببساطة. ننفذ عملية FIND-SET بتتبع مؤشرات الأب إلى أن نجد جذر الشجرة. تولّف العقد التي جرت زيارتها على هذا المسار البسيط إلى الجذر مسار الإيجاد find path. تتسبب عملية UNION الموضحة في الشكل 4.21(ب) في أن يقوم جذرُ إحدى الأشجار بالتأشير إلى جذر شجرة أخرى.



الشكل 4.21 غابة مجموعات منفصلة. (أ) شجرتان ثقلان المجموعتين في الشكل 2.21. تمثل الشجرة اليسارية المجموعة  $\{b, c, e, h\}$ ، حيث  $c$  هو الممثل، وتمثل الشجرة اليمينية المجموعة  $\{d, f, g\}$ ، حيث  $f$  هو الممثل. (ب) نتيجة الاجتماع  $UNION(e, g)$ .

### كسبيات لتحسين زمن التنفيذ

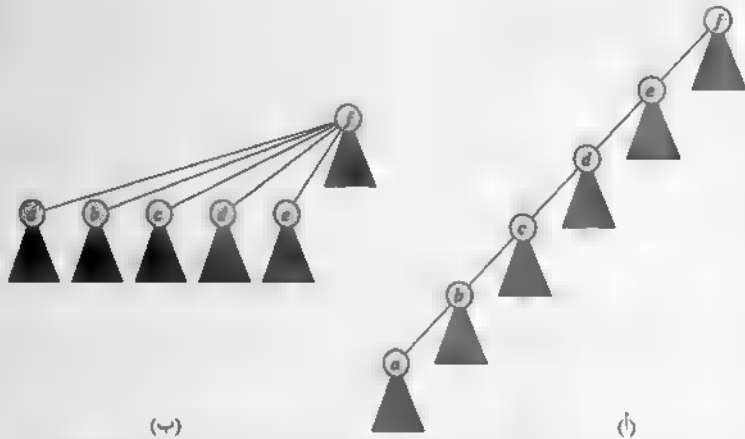
لم نحصل حتى الآن على تحسين لتعويض اللاتاحة للمقارنة. فمتتالية من  $n-1$  عملية UNION يمكن أن تُنشئ شجرة هي عبارة عن متتالية من  $n$  عقدة. ومع ذلك، يمكننا باستخدام كسبيتين الوصول إلى زمن تنفيذ خطي تقريباً نسبة إلى عدد العمليات  $m$ .

نشبه الكسبية الأولى (أي الاجتماع بحسب المرتبة *union by rank*) كسبية الاجتماع المثلث التي استخدمناها في تمثيل اللاتاحة المقارنة. النهج البديهي هو أن نجعل جذر الشجرة ذات العقد الأقل يشير إلى جذر الشجرة ذات العقد الأكثر. وبدلاً من الاحتفاظ بحجم الشجرة الفرعية في كل عقدة، سنستخدم هنا يُسهّل التحليل. سنحتفظ بمرتبة *rank* لكل عقدة هي حد أعلى لارتفاع العقدة. في الاجتماع بحسب المرتبة، نجعل الجذر ذا المرتبة الدنيا يشير إلى الجذر ذي المرتبة العليا خلال عملية UNION.

الكسبية الثانية (أي ضغط المسار *path compression*) هي أيضاً غاية في البساطة وفعالة جداً. وكما يظهر في الشكل 5.21، فإننا نستخدمها خلال عمليات FIND-SET لجعل كل عقدة في مسار الإيجاد تشير إلى الجذر مباشرة. لا يُغيّر ضغط المسار أية مرتبة.

### شبه رماز لغايات المجموعات المنفصلة

لتعويض غابة من المجموعات المنفصلة باستخدام كسبية الاجتماع بحسب المرتبة، لا بد أن نحفظ بالمراتب. في كل عقدة  $x$  نحافظ على القيمة الصحيحة  $rank.x$  التي هي حد أعلى لارتفاع  $x$  (عدد الوصلات في أطول مسار من ورقة متحلدة إلى  $x$ ). عندما يُنشئ MAKE-SET مجموعة وحيدة العنصر تكون مرتبة هذه العقدة الوحيدة 0. لا تُغيّر عملية FIND-SET من المراتب. تتضمن عملية UNION حالتين، اعتماداً على كون الجذرين متساويين في المرتبة أو لا. فإذا كانا غير متساويين في المرتبة، نجعل الجذر ذا المرتبة الأكبر أباً للجذر ذي المرتبة



الشكل 5.21 ضغط المسار خلال عملية FIND-SET. جرى حذف الأسهم والحلقات الذاتية في الجذور.  
 (أ) شجرة تمثل مجموعة قبل تنفيذ FIND-SET(a). تمثل الحلقات الأشجار الفرعية التي حلورها هي المقعد الظاهرة.  
 ولكل عقدة مؤشر إلى أبيها. (ب) المجموعة نفسها بعد تنفيذ FIND-SET(a). تشير كل عقدة على مسار الإيجاد الآن إلى الجذر مباشرة.

الأصغر دون أن نغير للترتيب. وإذا كانا متساويين في الترتيب، فإننا نختار أحدهما اعتباطيًا ليكون أبًا ونزيد رتبته بمقدار واحد.

لنضع هذه الطريقة في شبه رماز. نرمز للأب في عقدة ما  $x$  بـ  $x.p$ . الإجراء LINK هو مساق فرعي يستدعيه UNION، ويأخذ مؤشرين إلى عقدتين كمدخلين.

MAKE-SET(x)

- 1  $x.p = x$
- 2  $x.rank = 0$

UNION(x, y)

- 1 LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)

- 1 if  $x.rank > y.rank$
- 2  $y.p = x$
- 3 else  $x.p = y$
- 4 if  $x.rank == y.rank$
- 5  $y.rank = y.rank + 1$

الإجراء FIND-SET مع ضغط المسار هو إجراء بسيط جدًا.

**FIND-SET( $x$ )**

```

1  ■  $x \neq x.p$ 
2       $x.p = \text{FIND-SET}(x.p)$ 
3  return  $x.p$ 
    
```

الإجراء FIND-SET هو طريقة *بمرورين* *two-pass method*: فهو يقوم أثناء عودته بالمرور الأول إلى أعلى مسار الإيجاد لإيجاد الجذر، وعند نشر العودية يقوم بمرور ثاني راجعاً إلى أسفل مسار الإيجاد لتحديث كل عقدة بحيث تشير إلى الجذر مباشرةً. يبعد كل استدعاء للإجراء FIND-SET القيمة  $x.p$  في السطر 3. إذا كان  $x$  هو الجذر، فإن FIND-SET يتجاوز السطر 2 ويعيد بدلاً من ذلك  $x.p$  الذي هو  $x$  وهذه هي الحالة التي ينتهي فيها الصعود العودي. وإلا، يُنقذ السطر 2، ويعيد الاستدعاء العودي مع المتوسط  $x.p$  مؤشراً إلى الجذر. يحدث السطر 2 العقدة  $x$  تشير مباشرةً إلى الجذر، ويعيد السطر 3 هذا المؤشر.

### تأثير الكسبيات على زمن التنفيذ

يُحسن الاجتماع بحسب المرتبة وضغط المسار، كلٌّ منهما على حدة، زمن تنفيذ العمليات على غابات المجموعات المنفصلة، ويكون هذا التحسين أعظم عند استخدامهما معاً. فالاجتماع بحسب المرتبة وحده يعطي زمن تنفيذ  $O(m \lg n)$  (انظر التمرين 4.21)، وهذا الحد يُتحكَّم (انظر التمرين 3.21-3). ومع أننا لن نثبت ذلك هنا إلا أنه في حالة  $n$  عملية MAKE-SET (ومن ثم  $n-1$  عملية UNION على الأكثر) و  $f$  عملية FIND-SET، فإن ضغط المسار وحده يعطي زمناً تنفيذياً في أسوأ الحالات  $\Theta(n + f \cdot (1 + \log_{2+f/n} n))$ .

عندما نستخدم كلا من الاجتماع وضغط المسار يكون زمن التنفيذ في أسوأ الحالات  $O(m \alpha(n))$ ، حيث  $\alpha(n)$  هو دالة بطيئة النمو جداً، نعرفها في المقطع 4.21. في أي تطبيق يمكن تحويله لينة معطيات المجموعات المنفصلة  $\alpha(n) \leq 4$ ، لذا يمكننا النظر إلى زمن التنفيذ على أنه خطي نسبةً إلى  $m$  في جميع الحالات العملية. ومع ذلك، فالقول الجازم هو أنه فوق خطي. نُثبت هذا الحد الأعلى في المقطع 4.21.

### تمارين

#### 1-3.21

أعد حل التمرين 2-2.21 باستخدام غابة مجموعات منفصلة مع الاجتماع بحسب المرتبة وضغط المسار.

#### 2-3.21

اكتب إصداراً غير عودي من FIND-SET مع ضغط المسار.

#### 3-3.21

أعط متتالية من  $m$  عملية MAKE-SET و UNION و FIND-SET، بحيث تكون فيها  $n$  عملية MAKE-SET،

وحيث تستغرق زمناً  $\Omega(m \lg n)$  عندما نستخدم الاجتماع بحسب المرتبة فقط.

#### 4-3.21

افترض أننا نرغب بإضافة العملية  $\text{PRINT-SET}(x)$  التي تطبع جميع عناصر مجموعة  $x$  بأي ترتيب. بين كيف يمكننا إضافة واصفة وحيدة فقط لكل عقدة في غابة مجموعات منفصلة بحيث يستغرق  $\text{PRINT-SET}(x)$  زمناً خطياً مع عدد عناصر مجموعة  $x$  ولا تتغير أزمته التنفيذ للمقارب لبقية العمليات. افترض أننا نستطيع طباعة كل عنصر في المجموعة بزمن  $O(1)$ .

#### \* 5-3.21

بين أن أية متتالية من  $m$  عملية  $\text{MAKE-SET}$  و  $\text{FIND-SET}$  و  $\text{LINK}$  حيث تظهر جميع عمليات  $\text{LINK}$  قبل أية عملية  $\text{FIND-SET}$ ، تستغرق زمناً  $O(m)$  فقط، إذا استخدمنا كلاً من ضغط المسار والاجتماع بحسب المرتبة. ما الذي يجري في الحالة نفسها إذا استخدمنا كسبة ضغط المسار لوحدها؟

### \* 4.21 تحليل الاجتماع بحسب المرتبة وضغط المسار

وجدنا في المقطع 3.21 أن ضم الاجتماع بحسب المرتبة إلى ضغط المسار يستغرق زمن تنفيذ  $O(m \alpha(n))$  في حالة  $m$  عملية على المجموعات المنفصلة على  $n$  عنصراً. سنفحص في هذا المقطع الدالة  $\alpha$  لمعرفة مدى بطء نموها. ثم نثبت زمن التنفيذ هذا باستخدام طريقة الكمون في التحليل المتحد.

دالة سرعة النمو جداً ودالتها العكسية البطيئة النمو جداً

نعرف الدالة

$$A_k(j) = \begin{cases} j+1 & \text{if } k=0, \\ A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1. \end{cases}$$

للأعداد الصحيحة  $k \geq 0$  و  $j \geq 1$ ، حيث يُستخدم التعبير  $A_{k-1}^{(j+1)}(j)$  تدوين التكرار الدالي المعطى في المقطع 2.3. ويكون تحديداً  $A_{k-1}^{(0)}(j) = j$  و  $A_{k-1}^{(i-1)}(j) = A_{k-1}^{(i)}(j)$  لكل  $i \geq 1$ . سنعتبر للوسط  $k$  هو مستوى  $level$  الدالة  $A$ .

تزداد الدالة  $A_k(j)$  تماماً بازدياد  $j$  و  $k$  معاً. ولمعرفة مدى سرعة نمو هذه الدالة نحصل أولاً على تعبيرين من الشكل اللغلق لكل من  $A_1(j)$  و  $A_2(j)$ .

#### توطئة 2.21

إن  $A_1(j) = 2j + 1$  لأي عدد صحيح  $j \geq 1$ .

**البرهان** نستخدم أولاً الاستقراء على  $i$  لبيان أنَّ  $A_0^{(i)}(j) = j + i$ . في الحالة الأساسية لدينا  $A_0^{(0)}(j) = j = j + 0$ . نفترض للخطوة الاستقرائية أنَّ  $A_0^{(i-1)}(j) = j + (i - 1)$ . فيكون  $A_1^{(i)}(j) = A_0^{(i-1)}(j) = j + (i - 1)$ . نلاحظ أخيراً أنَّ  $A_0^{(i)}(j) = A_0(A_0^{(i-1)}(j)) = (j + (i - 1)) + 1 = j + i$ .  $j + (j + 1) = 2j + 1$  ■

### 3.21 توطئة

إن  $A_2(j) = 2^{j+1}(j + 1) - 1$  لأي عدد صحيح  $j \geq 0$ .

**البرهان** نستخدم أولاً الاستقراء على  $i$  لبيان أنَّ  $A_1^{(i)}(j) = 2^i(j + 1) - 1$ . في الحالة الأساسية لدينا  $A_1^{(0)}(j) = j = 2^0(j + 1) - 1$ . نفترض للخطوة الاستقرائية أنَّ  $A_1^{(i-1)}(j) = 2^{i-1}(j + 1) - 1$ . فيكون

$$\begin{aligned} A_1^{(i)}(j) &= A_1(A_1^{(i-1)}(j)) = A_1(2^{i-1}(j + 1) - 1) = 2 \cdot (2^{i-1}(j + 1) - 1) + 1 \\ &= 2^i(j + 1) - 2 + 1 = 2^i(j + 1) - 1 \end{aligned}$$

نلاحظ أخيراً أنَّ  $A_2(j) = A_1^{(j+1)}(j) = 2^{j+1}(j + 1) - 1$  ■

يمكننا الآن ملاحظة مدى سرعة نمو  $A_k(j)$  بتفحص  $A_k(1)$  للمستويات  $k = 0, 1, 2, 3, 4$  فقط. من تعريف  $A_0(k)$  والتوطينتين السابقتين لدينا  $A_0(1) = 1 + 1 = 2$  و  $A_1(1) = 2 \cdot 1 + 1 = 3$  و  $A_2(1) = 2^{1+1} \cdot (1 + 1) - 1 = 7$  ولدينا أيضاً:

$$\begin{aligned} A_3(1) &= A_2^{(2)}(1) \\ &= A_2(A_2(1)) \\ &= A_2(7) \\ &= 2^8 \cdot 8 - 1 \\ &= 2^{11} - 1 \\ &= 2047 \end{aligned}$$

و

$$\begin{aligned} A_4(1) &= A_3^{(2)}(1) \\ &= A_3(A_3(1)) \\ &= A_3(2047) \\ &= A_2^{(2048)}(2047) \\ &\gg A_2(2047) \\ &= 2^{2048} \cdot 2048 - 1 \\ &> 2^{2048} \end{aligned}$$

$$\begin{aligned}
 &= (2^4)^{512} \\
 &= 16^{512} \\
 &\gg 10^{80} ,
 \end{aligned}$$

وهو العدد المتوقع للذرات في الكون المنظور. (مثل الرمز «علاقة» أكبر بكثير من «).  
نعرف الدالة المعاكسة للدالة  $A_k(n)$  لكل عدد صحيح  $n \geq 0$  كما يلي:

$$\alpha(n) = \min\{k : A_k(1) \geq n\}$$

باختصار  $\alpha(n)$  هو المستوى الأدنى  $k$  الذي يكون من أجله  $A_k(1)$  هو  $n$  على الأقل. نرى من قيم  $A_k(1)$  المذكورة آنفاً أنَّ

$$\alpha(n) = \begin{cases} \infty & \text{for } 0 \leq n \leq 2 , \\ 1 & \text{for } n = 3 , \\ 2 & \text{for } 4 \leq n \leq 7 , \\ 3 & \text{for } 8 \leq n \leq 2047 , \\ 4 & \text{for } 2048 \leq n \leq A_4(1) . \end{cases}$$

أي تكون  $\alpha(n) > 4$  فقط لقيم كبيرة جداً للدرجة "فلكية" لـ  $n$  (أكبر من  $A_4(1)$ ، رقم هائل جداً)، وبذلك تكون  $\alpha(n) \leq 4$  لجميع الأغراض العملية.

#### خصائص المراتب

نُثبت فيما بقي من هذا المقطع أنَّ  $O(m\alpha(n))$  هي حدٌ لزمين التنفيذ في العمليات على المجموعات المنفصلة باستخدام الاجتماع بحسب للترتبة وضغط المسار. بهدف إثبات هذا الحد، نبدأ أولاً بإثبات بعض خصائص للمراتب.

#### توطئة 4.21

إن  $x.rank \leq x.p.rank$  مع مراجعة تامة إذا كان  $x \neq x.p$ ، وذلك لجميع العقد  $x$ . تكون قيمة  $x.rank$  في البداية 0 وتزداد مع الوقت إلى أن يصبح  $x \neq x.p$ ؛ ولا تتغير بعد ذلك قيمة  $x.rank$ . تزداد قيمة  $x.p.rank$  باطراد مع مرور الزمن.

**البرهان** نبرهن هذه التوطئة بالاستقراء المباشر على عدد العمليات باستخدام تنحييزات MAKE-SET و UNION و FIND-SET التي تظهر في للمقطع 3.21. مشترك البرهان للتمرين 1-4.21.

#### نتيجة 5.21

عندما تتبع المسار البسيط من أية عقدة إلى الجذر، فإنَّ مراتب العقد تزايد تماثلاً.



### توطئة 6.21

مرتبة كل عقدة هي  $n-1$  على الأكثر.

**البرهان** نبدأ مرتبة كل عقدة من 0 وتزداد في عمليات LINK فقط. وبسبب وجود  $n-1$  عملية UNION على الأكثر، توجد  $n-1$  عملية LINK على الأكثر. ولما كانت كل عملية LINK إما أن تُبقي المراتب على ما هي عليه وإما أن تزيد مراتب بعض العقد بمقدار 1، فإن كل المراتب هي  $n-1$  على الأكثر. ■

توفر التوطئة 6.21 حدًا ضمنيًا على المراتب. والحقيقة أن مرتبة كل عقدة هي  $\lg n$  على الأكثر (انظر التمرين 2-4.21). ومع ذلك، فإن الحد المنحل للتوطئة 6.21 سيكفي لتحقيق أهدافنا.

### إثبات الحد الزمني

سنستخدم طريقة الكمون في التحليل المصحّد (انظر للمقطع 3.17) لإثبات الحد الزمني  $O(m \alpha(n))$ . عند إجراء تحليل محصّد سنجد أن من المناسب أن نفترض أننا نستدعي عملية LINK بدلاً من عملية UNION. وذلك لأنّ موسطي الإجراء LINK هما مؤشران إلى جذرين، بافتراض أننا نُنجز عمليات FIND-SET بصورة مستقلة. تُبيّن التوطئة التالية أننا حتى لو أخذنا بالمسبان عمليات FIND-SET الإضافية التي تنتج عن استدعاءات UNION، فإن زمن التنفيذ المقارب لا يتغيّر.

### توطئة 7.21

افترض أننا نحول متتالية  $S'$  من  $m'$  عملية MAKE-SET و UNION و FIND-SET إلى متتالية  $S$  من  $m$  عملية MAKE-SET و LINK و FIND-SET بتحويل كل UNION إلى عمليتي FIND-SET تتلوها عملية LINK. عندئذٍ، إذا كانت المتتالية  $S$  تُنفّذ بزمن  $O(m \alpha(n))$ ، فإنّ للمتتالية  $S'$  تُنفّذ بزمن  $O(m' \alpha(n))$ .

**البرهان** لما كانت كل عملية UNION في المتتالية  $S'$  تُحوّل إلى ثلاث عمليات في  $S$ ، فلدينا  $m' \leq m \leq 3m'$ . ولما كان  $m = O(m')$ ، فإنّ الحد الزمني للمتتالية المحوّلة  $O(m \alpha(n))$  يقتضي حدًا زمنيًا  $O(m' \alpha(n))$  للمتتالية الأصلية  $S'$ . ■

سنفترض فيما تبقى من هذا المقطع أنّ للمتتالية البدائية المولفة من  $m'$  عملية MAKE-SET و UNION و FIND-SET قد جرى تحويلها إلى متتالية من  $m$  عملية MAKE-SET و LINK و FIND-SET. وثبت الآن حدًا زمنيًا  $O(m \alpha(n))$  للمتتالية المحوّلة، ونعتمد على التوطئة 7.21 لإثبات زمن تنفيذ  $O(m' \alpha(n))$  للمتتالية الأصلية ذات الـ  $m'$  عملية.

## دالة الكمون

تسند دالة الكمون التي نستخدمها كمونًا  $\phi_q(x)$  لكل عقدة  $x$  في غابة المجموعات المنفصلة بعد  $q$  عملية. نجمع كمون العقد لنحصل على كمون الغاية كاملة:  $\Phi_q = \sum_x \phi_q(x)$ ، حيث يرمز  $\Phi_q$  إلى كمون الغاية بعد  $q$  عملية. تكون الغاية فارغة قبل العملية الأولى، ونختار  $\Phi_0 = 0$ . ولن يكون أي كمون  $\Phi_q$  سالبًا أبدًا. تعتمد قيمة  $\phi_q(x)$  على كون  $x$  جذرًا للشجرة بعد العملية  $q$ . فإذا كان كذلك، أو كان  $x.rank = 0$ ، فإن  $\phi_q(x) = \alpha(n) \cdot x.rank$ .

افترض الآن أنه بعد العملية  $q$  لن يكون  $x$  جذرًا، وأن  $x.rank \geq 1$ . نحتاج إلى تعريف دالتين مساعدتين على  $x$  قبل أن نستطيع تعريف  $\phi_q(x)$ . نعرف أولاً

$$level(x) = \max \{k : x.p.rank \geq A_k(x.rank)\} .$$

أي إنَّ  $level(x)$  هو المستوى الأعظم  $k$  الذي لا يكون من أجله  $A_k$  المطبق على مرتبة  $x$  أكبر من مرتبة أبي  $x$ .

إنَّ

$$0 \leq level(x) < \alpha(n) , \quad (1.21)$$

التي ننظر إليها كما يلي. لدينا

$$x.p.rank \geq x.rank + 1 \quad ((4.21) \text{ بحسب التوطئة})$$

$$= A_0(x.rank) , \quad (A_0(f) \text{ تعريف})$$

وهذا يقتضي أنَّ  $level(x) \geq 0$  ويكون

$$A_{\alpha(n)}(x.rank) \geq A_{\alpha(n)}(1) \quad (\text{لأن } A_k(f) \text{ متزايد تمامًا})$$

$$\geq n \quad (\alpha(n) \text{ بحسب تعريف})$$

$$> x.p.rank , \quad ((6.21) \text{ بحسب التوطئة})$$

وهذا بدوره يقتضي أن يكون  $level(x) < \alpha(n)$ . لاحظ أنه لما كان  $x.p.rank$  متزايدًا باطراد مع الزمن، فإنَّ  $level(x)$  هو متزايد باطراد أيضًا.

تُطبَّق الدالة للمساعدة الثانية عندما يكون  $x.rank \geq 1$

$$iter(x) = \max \{i : x.p.rank \geq A_{level(x)}^{(i)}(x.rank)\} .$$

أي إنَّ  $iter(x)$  هو أكبر عدد من المرات التي يمكننا فيها تطبيق  $A_{level(x)}$  للمطبَّق بدايةً على مرتبة  $x$ ، قبل أن نحصل على قيمة أكبر من مرتبة أبي  $x$ .

عندما تكون  $x.rank \geq 1$  يكون لدينا:

$$1 \leq \text{iter}(x) \leq x.rank, \quad (2.21)$$

التي ننظر إليها كما يلي. لدينا

$$\begin{aligned} x.p.rank &\geq A_{\text{level}(x)}(x.rank) && (\text{بحسب تعريف } \text{level}(x)) \\ &= A_{\text{level}(x)}^{(1)}(x.rank) && (\text{بحسب تعريف التكرار الوظيفي}) \end{aligned}$$

وهذا يقتضي أنَّ  $\text{iter}(x) \geq 1$  ويكون لدينا

$$\begin{aligned} A_{\text{level}(x)}^{(x.rank+1)}(x.rank) &= A_{\text{level}(x)+1}(x.rank) && (\text{بحسب تعريف } A_k(j)) \\ &> x.p.rank, && (\text{بحسب تعريف } \text{level}(x)) \end{aligned}$$

وهذا بدوره يقتضي أنَّ  $\text{iter}(x) \leq x.rank$ . لاحظ أنه لئلا كان  $x.p.rank$  متزايد باطراد مع الزمن (لكي يتناقص  $\text{iter}(x)$ )، فإن  $\text{level}(x)$  يجب أن يتزايد. ومع بقاء  $\text{level}(x)$  دون تغيير، فإنَّ  $\text{iter}(x)$  إما أن يتزايد وإما أن يبقى دون تغيير.

بعد تعريف هاتين الدالتين، نعرف كمون عقدة  $x$  بعد  $q$  عملية:

$$\phi_q(x) = \begin{cases} \alpha(n) \cdot x.rank & \text{if } x \text{ is a root or } x.rank = 0, \\ (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) & \text{if } x \text{ is not a root and } x.rank \geq 1. \end{cases}$$

تعطي التوطلتان التاليتان خواص مفيدة لكمونات العقد.

### 8.21 توطئة

$$0 \leq \phi_q(x) \leq \alpha(n) \cdot x.rank$$

لكل عقدة  $x$  ولكل أعداد العمليات  $q$ .

**البرهان** إذا كان  $x$  جذراً أو كان  $x.rank = 0$ ، فإن  $\phi_q(x) = \alpha(n) \cdot x.rank$ . افترض الآن أنَّ  $x$  ليس جذراً وأنَّ  $x.rank \geq 1$ . نحصل على حد أدنى لـ  $\phi_q(x)$  بتعظيم قيم  $\text{level}(x)$  و  $\text{iter}(x)$ . وبحسب الحد (1.21)، يكون  $\text{level}(x) \leq \alpha(n) - 1$ ، وبحسب الحد (2.21)، يكون  $\text{iter}(x) \leq x.rank$  وبذلك يكون

$$\begin{aligned} \phi_q(x) &= (\alpha(n) - \text{level}(x)) \cdot x.rank - \text{iter}(x) \\ &\geq (\alpha(n) - (\alpha(n) - 1)) \cdot x.rank - x.rank \\ &= x.rank - x.rank \\ &= 0. \end{aligned}$$

وبمثل نحصل على حد أعلى على  $\phi_q(x)$  بتصغير قيم  $\text{level}(x)$  و  $\text{iter}(x)$ . وبحسب الحد (1.21)، يكون

$level(x) \geq 0$ ، وبحسب الحد (2.21) يكون  $iter(x) \geq 1$ ، وبذلك يكون

$$\begin{aligned}\phi_q(x) &\leq (\alpha(n) - 0) \cdot x.rank - 1 \\ &= \alpha(n) \cdot x.rank - 1 \\ &< \alpha(n) \cdot x.rank.\end{aligned}$$

### نتيجة 9.21

إذا لم تكن العقدة جذراً، وكان  $x.rank > 0$ ، فإن  $\phi_q(x) < \alpha(n) \cdot x.rank$ .

### تغيرات الكمون والتكلفة المخمّدة للعمليات

لندرس الآن تأثير عمليات المجموعات المنفصلة على كمونات العقد. إن معرفة التغير في الكمون نتيجة كل عملية، يمكننا من تحديد التكلفة المخمّدة لها.

### توطئة 10.21

لنكن  $x$  عقدة ليست جذراً، ولنفترض أنّ العملية  $q$  هي LINK أو FIND-SET. فيكون لدينا بعد العملية ذات الرقم  $q$   $\phi_q(x) \leq \phi_{q-1}(x)$ . وإذا كان  $x.rank \geq 1$ ، وتغير  $level(x)$  أو  $iter(x)$  بسبب العملية  $q$ ، فإن  $\phi_q(x) \leq \phi_{q-1}(x) - 1$ . أي إنّ كمون  $x$  لا يمكن أن يزداد، وإذا كان له مرتبة موجبة وتغير  $level(x)$  أو  $iter(x)$ ، فإنّ كمون  $x$  ينقص بمقدار 1 على الأقل.

**البرهان** إن  $x$  ليست جذراً، لذا فإنّ العملية ذات الرقم  $q$  لا تغير  $x.rank$ ، ولما كانت  $n$  لا تتغير بعد  $n$  عملية MAKE-SET بادئها، فإنّ  $\alpha(n)$  يبقى أيضاً دون تغيير، لأن هذه المركبات في صيغة كمون  $x$  تبقى نفسها بعد العملية  $q$ . فإذا كان  $x.rank = 0$ ، فإن  $\phi_q(x) = \phi_{q-1}(x) = 0$ . افترض الآن أنّ  $x.rank \geq 1$ .

تذكر أنّ  $level(x)$  تزداد باطراد مع الزمن. فإذا تركت العملية ذات الرقم  $q$  القيمة  $level(x)$  دون تغيير، ازداد  $iter(x)$  أو بقي دون تغيير. وإذا لم يتغير أيّ من  $level(x)$  و  $iter(x)$ ، فإن  $\phi_q(x) = \phi_{q-1}(x)$ . وإذا لم يتغير  $level(x)$  وازداد  $iter(x)$ ، فإنه يزداد بمقدار 1 على الأقل، وبذلك يكون  $\phi_q(x) \leq \phi_{q-1}(x) - 1$ .

أخيراً، إذا زادت العملية ذات الرقم  $q$  من  $level(x)$ ، فإنه يزداد بمقدار 1 على الأقل، أي إنّ قيمة الحد  $(\alpha(n) - level(x)) \cdot x.rank$  تنخفض بمقدار  $x.rank$  على الأقل. ولما كان  $level(x)$  قد ازداد، فإنّ قيمة  $iter(x)$  قد تنخفض، لكن بحسب الحد (2.21)، يكون الانخفاض بمقدار  $x.rank - 1$ . أي إنّ زيادة الكمون نتيجة لتغير  $iter(x)$  هي أقل من نقصان الكمون نتيجة لتغير  $level(x)$ ، ونستنتج أنّ  $\phi_q(x) \leq \phi_{q-1}(x) - 1$ .

تبيّن التوططات الثلاث الأخيرة أنّ الكمون المختد لكل عملية من العمليات MAKE-SET و LINK و FIND-SET هو  $O(\alpha(n))$ . تدكر من المعادلة (2.17) أنّ التكلفة المختدة لكل عملية هي تكلفتها الفعلية يُضاف إليها الزيادة في الكمون نتيجة للعملية.

### توططة 11.21

التكلفة المختدة لكل عملية MAKE-SET هي  $O(1)$ .

**البرهان** افترض أنّ العملية  $q$  هي  $MAKE-SET(x)$ . تنشئ هذه العملية عقدة  $x$  مرتبتها 0، أي  $\phi_q(x) = 0$ ، دون أن تتغير بقية المراتب أو الكمونات، وبذلك يكون  $\phi_q = \phi_{q-1}$ . وملاحظة أنّ التكلفة الفعلية لعملية MAKE-SET هي  $O(1)$  يكتمل البرهان. ■

### توططة 12.21

التكلفة المختدة لكل عملية LINK هي  $O(\alpha(n))$ .

**البرهان** افترض أنّ العملية  $q$  هي  $LINK(x, y)$ . التكلفة الفعلية لعملية LINK هي  $O(1)$ . دون فقد للعمومية، افترض أنّ عملية LINK تجعل  $y$  أبنا لـ  $x$ .

لتحديد التغير في الكمون بنتيجة عملية LINK، نلاحظ أنّ العقد الوحيدة التي يمكن أن تتغير هي  $x$  و  $y$  وأبناء  $y$  قبل العملية مباشرة. سنبيّن أنّ العقدة الوحيدة التي يمكن أن يزيد كمونها بنتيجة عملية LINK هي  $y$ ، وأنّ هذه الزيادة هي على الأكثر  $\alpha(n)$ :

- بحسب التوططة 10.21 فإنّ أي عقدة كانت ابناً لـ  $y$  قبل عملية LINK مباشرة لا يمكن أن يزداد كمونها بنتيجة العملية LINK.

- من تعريف  $\phi_q(x)$  نرى أنّه لما كان  $x$  جذراً قبل العملية  $q$  مباشرة فإنّ  $\phi_{q-1}(x) = \alpha(n) \cdot x.rank$  فإذا كان  $x.rank = 0$ ، فإن  $\phi_q(x) = \phi_{q-1}(x) = 0$ ، وإلا فإن

$$\phi_q(x) < \alpha(n) \cdot x.rank \quad (\text{بحسب النتيجة (9.21)})$$

$$= \phi_{q-1}(x),$$

ومن ثمّ فإنّ كمون  $x$  ينقص.

- لما كان  $y$  جذراً قبل عملية LINK، فإنّ  $\phi_{q-1}(y) = \alpha(n) \cdot y.rank$ . وتبقى عملية LINK  $y$  جذراً، ثمّ إما أن تبقى مرتبة  $y$  على ما هي عليه، وإما أن تزيد بمقدار 1. لذا فإن  $\phi_q(y) = \phi_{q-1}(y) + \alpha(n)$  أو  $\phi_q(y) = \phi_{q-1}(y)$ .

فالزيادة في الكمون بتنتيجة العملية LINK هي إذن  $\alpha(n)$  على الأكثر. والتكلفة للخطوة لعملية LINK هي  $O(1) + \alpha(n) = O(\alpha(n))$ .

### 13.21 توطئة

التكلفة للخطوة لكل عملية FIND-SET هي  $O(\alpha(n))$ .

**البرهان** افترض أن العملية  $q$  هي FIND-SET وأن مسار الإيجاد يحتوي  $s$  عقدة. إن التكلفة الفعلية لعملية FIND-SET هي  $O(s)$ . سنبين أن كمون العقد لا يزداد بتنتيجة عملية FIND-SET وأن هناك  $\max(0, s - (\alpha(n) + 2))$  عقدة على الأقل على مسار الإيجاد ينقص كمونها بمقدار 1 على الأقل. لبيان أن كمون العقد لا يزداد، نلجأ أولاً إلى التوطئة 10.21 لجميع العقد ما عدا الجذر. إذا كان  $x$  هو الجذر، فإن كمونه هو  $\alpha(n) \cdot x.rank$ ، وهو لا يتغير.

نبين الآن أن هناك  $\max(0, s - (\alpha(n) + 2))$  عقدة على الأقل ينقص كمونها بمقدار 1 على الأقل. لتكن  $x$  عقدة على مسار الإيجاد بحيث  $x.rank > 0$  وتتبعها في مكان ما من مسار الإيجاد عقدة أخرى  $y$  ليست جذراً، بحيث يكون  $level(y) = level(x)$  قبل عملية FIND-SET مباشرة. (لاحظ أنه ليس بالضرورة أن تكون العقدة  $y$  تتبع العقدة  $x$  مباشرة على مسار الإيجاد.) جميع العقد على مسار الإيجاد عدا  $\alpha(n) + 2$  عقدة تحقق هذه القيود على  $x$ . فالعقد التي لا تحققها هي العقدة الأولى على مسار الإيجاد (إذا كانت مرتبتها 0)، والعقدة الأخيرة على المسار (أي الجذر)، والعقدة الأخيرة  $w$  على المسار التي يكون عندها  $level(w) = k$  في حال  $k = 0, 1, 2, \dots, \alpha(n) - 1$ .

نثبت هذه العقدة  $x$ ، ونبين أن كمونها ينقص بمقدار 1 على الأقل. ليكن  $k = level(x) = level(y)$ . يكون لدينا قبل ضغط المسار الذي تسببه FIND-SET مباشرة

$$x.p.rank \geq A_k^{iter(x)}(x.rank) \quad (\text{بحسب تعريف } iter(x))$$

$$y.p.rank \geq A_k(y.rank) \quad (\text{بحسب تعريف } iter(y))$$

$$y.rank \geq x.p.rank \quad (\text{بحسب النتيجة (5.21) ولأن } y \text{ تتبع } x \text{ على مسار الإيجاد})$$

بوضع هذه المتراجحات معاً، وبافتراض أن  $i$  هي قيمة  $iter(x)$  قبل ضغط المسار، يكون لدينا

$$\begin{aligned} y.p.rank &\geq A_k(y.rank) \\ &\geq A_k(x.p.rank) \quad (\text{لأن } A_k(f) \text{ متزايد تماماً}) \end{aligned}$$

$$\geq A_k(A_k^{iter(x)}(x.rank))$$

$$= A_k^{i+1}(x.rank).$$

لما كان ضغط المسار سيجعل لكل من  $x$  و  $y$  الأب نفسه، فإننا نعلم أنه بعد ضغط المسار سيكون  $x.p.rank = y.p.rank$  وأن ضغط المسار لا يُقَصِّص  $y.p.rank$ . ولما كان  $x.rank$  لا يتغير بعد ضغط المسار فلدينا  $x.p.rank \geq A_k^{l+1}(x.rank)$ . وهكذا، فإن ضغط المسار سيسبب ازدياد  $iter(x)$  إلى  $l+1$  على الأقل. في كلا الحالتين، وبحسب التوطئة 10.21، يكون لدينا  $1 \sim \phi_q(x) \leq \phi_{q-1}(x)$ . ومن ثم فإن كمون  $x$  ينقص بمقدار 1 على الأقل.

إن التكلفة المَحْتَمَدة لعملية FIND-SET هي التكلفة الفعلية إضافة إلى التغير في الكمون. والتكلفة الفعلية هي  $O(s)$ ، وقد يَبْدُو أنَّ الكمون الكلي ينقص بمقدار  $\max(0, s - (\alpha(n) + 2))$ . وبذلك تكون التكلفة المَحْتَمَدة هي على الأكثر  $O(\alpha(n)) = O(\alpha(n)) = O(s) - s + O(\alpha(n) + 2) = O(s) - (s - (\alpha(n) + 2))$ . لأننا نستطيع رفع وحدات الكمون لتغطي على الثابت المضمر في  $O(s)$ .

يَتَبَعُ عن التوطئات السابقة جميعها البرهنة التالية.

#### مبرهنة 14.21

يمكن تنفيذ متتالية من  $m$  عملية MAKE-SET و UNION و FIND-SET من بينها  $n$  عملية MAKE-SET على غابة من المجموعات المنفصلة باستخدام الاجتماع المَثْقَل وضغط المسار في أسوأ الحالات بزمن  $O(m \alpha(n))$ .

**البرهان** يتحقق البرهان مباشرة من التوطئات 7.21 و 11.21 و 12.21 و 13.21.

#### تمارين

##### 1-4.21

أثبت التوطئة 4.21.

##### 2-4.21

أثبت أنَّ مرتبة أية عقدة هي  $\lg n$  على الأكثر.

##### 3-4.21

على ضوء التمرين 2-4.21، كم عدد الثبات الضرورية لحزن  $x.rank$  لكل عقدة  $x$ ؟

##### 4-4.21

باستخدام التمرين 2-4.21، أعطِ برهاناً بسيطاً على أنَّ العمليات على غابة المجموعات المنفصلة مع الاجتماع بحسب المرتبة ولكن بدون ضغط المسار تُنفَّذ بزمن  $O(m \lg n)$ .

#### 5-4.21

يعتقد الأستاذ Dante أنه لما كانت مراتب العقد تتزايد تمامًا على مسار بسيط إلى الجذر، فإن مستويات العقد يجب أن تتزايد باطراد على ذلك المسار. بتعبير آخر، إذا كان  $x.rank > 0$  ولم يكن  $x.p$  جذرًا، فإن  $level(x) \leq level(x.p)$ . هل الأستاذ على صواب؟

#### \* 6-4.21

لتكن لدينا الدالة  $\alpha'(n) = \min \{k : A_k(1) \geq \lg(n+1)\}$ . بين أن  $\alpha'(n) \leq \alpha(n)$  لجميع القيم العملية لـ  $n$ ، وبين باستخدام التمرين 2-4.21 كيف يمكن تعديل عدد دالة الكمون لإثبات أنه يمكننا تنفيذ متتالية من  $m$  عملية MAKE-SET و UNION و FIND-SET من بينها  $n$  عملية MAKE-SET على غاية من المجموعات المنفصلة باستخدام الاحتماع بحسب المرتبة وضغط المسار في أسوأ الحالات بزم  $O(m \alpha'(n))$ .

### مسائل

#### 1-21 الحد الأصغر خارج الخط

المطلوب في مسألة الحد الأصغر خارج الخط *off-line minimum problem* أن نحافظ على مجموعة ديناميكية  $T$  من العناصر من المجال  $\{1, 2, \dots, n\}$  مع العمليات INSERT و EXTRACT-MIN. لدينا متتالية  $S$  من  $n$  استدعاء INSERT و  $m$  استدعاء EXTRACT-MIN، حيث يجري إدراج كل مفتاح من  $\{1, 2, \dots, n\}$  مرة واحدة. نرغب بتحديد المفتاح الذي يعيده كل استدعاء EXTRACT-MIN. ونرغب تحديدًا بملاء صيغة  $extracted[1..m]$ ، حيث  $extracted[i]$  هو للمفتاح الذي يعيده الاستدعاء  $i$  لـ EXTRACT-MIN لكل  $i = 1, 2, \dots, m$ . تعني مسألة "خارج الخط" أنه يمكننا معالجة للمتتالية  $S$  كاملة قبل تحديد أي من المفاتيح المعادة.

أ. في المتسخ (instance) التالي من مسألة الحد الأصغر خارج الخط، تمثل كل عملية INSERT( $i$ ) بقيمة  $i$  وتمثل كل EXTRACT-MIN بحرف E:

4, B, E, 3, E, 9, 2, 6, E, E, 1, 7, E, 5.

املأ القيم الصحيحة في الصيغة *extracted*.

لإنشاء خوارزمية لهذه المسألة، نقسم للمتتالية  $S$  إلى متتاليات جزئية متجانسة. أي نقل  $S$  كما يلي:

$I_1, E, I_2, E, I_3, \dots, I_m, E, I_{m+1}, \dots$

حيث تمثل كل  $E$  استدعاء واحدًا لـ EXTRACT-MIN وتمثل كل  $I_i$  متتالية من استدعاءات INSERT (قد تكون فارغة). نضع في البداية لكل متتالية جزئية  $I_i$  للمفاتيح للدرجة في هذه العمليات ضمن مجموعة  $K_i$  (تكون فارغة إذا كانت  $I_i$  فارغة). ثم نقوم بما يلي:



OFF-LINE-MINIMUM( $m, n$ )

```

1  for  $i = 1$  to  $n$ 
2      determine  $j$  such that  $i \in K_j$ 
3      if  $j \neq m + 1$ 
4           $extracted[j] = i$ 
5          let  $l$  be the smallest value greater than  $j$ 
              for which set  $K_l$  exists
6           $K_l = K_j \cup K_l$ , destroying  $K_j$ 
7  return  $extracted$ 

```

ب. ناقش صحة كون الصنيفة  $extracted$  التي يعيدها OFF-LINE-MINIMUM صحيحة.

ت. صِفْ كيفية تنجيز OFF-LINE-MINIMUM تنجيزاً فعالاً مع بنية معطيات للمجموعات المنفصلة. أعطِ حداً تُحَكِّمُهُ الزمن تنفيذ تنجيزك في أسوأ الحالات.

## 2-21 تحديد العمق

في مسألة تحديد العمق  $depth-determination$  نحافظ على غابة  $\mathcal{F} = \{T_i\}$  من الأشجار ذات الجذور مع ثلاث عمليات:

MAKE-TREE( $v$ ) إنشاء شجرة ذات عقدة واحدة  $v$ .

FIND-DEPTH( $v$ ) إعادة عمق عقدة  $v$  ضمن شجرتها.

GRAFT( $r, v$ ) يُعَلِّق العقدة  $r$  (التي يفترض أن تكون هي جذر الشجرة) ابناً لعقدة  $v$  (التي يفترض أن تكون في شجرة أخرى غير  $r$  ولكن يمكن أن تكون جذراً أو لا).

أ. افترض أننا نستخدم تمثيلاً للشجرة يشبه غابة المجموعات المنفصلة:  $v.p$  هو أب للعقدة  $v$ ، إلا إذا كان  $v$  جذراً فيكون  $v.p = v$ . افترض أيضاً أننا نجزنا GRAFT( $r, v$ ) بوضع  $r.p = v$  و FIND-DEPTH( $v$ ) باتباع مسار الإيجاد إلى الجذر، وإعادة عدد العقد غير  $v$  التي جرت مصادفتها، بيّن أنّ زمن التنفيذ في أسوأ الحالات لمتتالية من  $m$  عملية MAKE-TREE و FIND-SET و GRAFT هو  $\Theta(m^2)$ .

باستخدام كسبيتي الاحتماع بحسب لمرتبة وضغط المسار يمكننا تقليص زمن التنفيذ في أسوأ الحالات. نستخدم غابة المجموعات المنفصلة  $\mathcal{S} = \{S_i\}$ ، حيث تقابل كل مجموعة  $S_i$  (التي هي شجرة بحد ذاتها) شجرة  $T_i$  في الغابة  $\mathcal{F}$ . ومع ذلك، فإن بنية الشجرة الخاصة بالمجموعة  $S_i$  لا تقابل بالضرورة بنية المعطيات الخاصة بـ  $T_i$ . في الحقيقة لا يُسَجَّل تنجيز  $S_i$  العلاقة الدقيقة بين الأب والابن، ومع ذلك فهو يسمح بتحديد عمق أية عقدة في  $T_i$ .

الفكرة الرئيسية هي الحفاظ على "شبه مسافة"  $v.d$  في كل عقدة  $v$ ، التي تعرّف بحيث يكون مجموع أشباه المسافات على المسار البسيط من  $\square$  إلى جذر مجموعتها  $S_v$  مساويًا لعمق  $v$  في  $T_v$  أي إذا كان المسار البسيط من  $v$  إلى جذرها في  $S_v$  هو  $v_0, v_1, \dots, v_k$  حيث  $v_0 = v$  و  $v_k$  هو جذر  $S_v$ ، فيكون عمق  $v$  في  $T_v$  هو  $\sum_{j=0}^{k-1} v_j.d$ .

ب. أعطِ تنحيزًا لـ MAKE-TREE.

ت. بَيِّن كيف نعدّل FIND-SET لتنحيز FIND-DEPTH. يجب أن يُضبط تنحيزك للمسار وأن يكون زمن تنفيذه خطيًا بدلالة طول مسار الإيجاد. تأكد أن تنحيزك يُحدث أشباه المسافات تحديثًا صحيحًا.

ث. بَيِّن كيفية تنحيز  $\text{GRAFT}(r, v)$  الذي يجمع المجموعتين اللتضميتين  $r$  و  $v$  بتعديل الإجراءات UNION و LINK. تأكد أن تنحيزك يُحدث أشباه المسافات تحديثًا صحيحًا. لاحظ أن جذر مجموعة  $S_v$  ليس بالضرورة جذرًا للشجرة المقابلة  $T_v$ .

ج. أعطِ حدًا مُحكَّمًا لزمن التنفيذ في أسوأ الحالات لمتالية من  $m$  عملية MAKE-TREE و FIND-DEPTH و  $n$  عملية GRAFT من بينها.

### 2-21 خوارزمية تاريان خارج الخط للبحث عن السلف المشترك الأبعد

السلف المشترك الأبعد  $\text{least common ancestor}$  لعقدتين  $u$  و  $v$  في شجرة ذات جذر  $T$  هو العقدة  $w$  التي تكون سلفًا لكل من  $u$  و  $v$  ويكون لها العمق الأكبر في  $T$ . في مسألة الأسلاف المشتركة الأبعد خارج الخط  $\text{off-line least common-ancestors problem}$ ، لدينا شجرة  $T$  وبمجموعة  $P = \{(u, v)\}$  من الأزواج غير المرتبة من العقد في  $T$ ، ونرغب بتحديد السلف المشترك الأبعد لكل زوج في  $P$ .  
لحل مسألة الأسلاف المشتركة الأبعد خارج الخط، يقوم الإجراء التالي بتحويل  $T$  مع استدعاء ابتدائي  $\text{LCA}(T, \text{root})$ . نفترض تلوين كل عقدة بالأبيض WHITE قبل التحويل.

$\text{LCA}(u)$

- 1 MAKE-SET( $u$ )
- 2 FIND-SET( $u$ ).ancestor =  $u$
- 3 for each child  $v$  of  $u$  in  $T$
- 4     LCA( $v$ )
- 5     UNION( $u, v$ )
- 6     FIND-SET( $u$ ).ancestor =  $u$
- 7  $u.\text{color} = \text{BLACK}$
- 8 for each node  $v$  such that  $\{u, v\} \in P$
- 9     if  $v.\text{color} == \text{BLACK}$
- 10         print "The least common ancestor of"  
            $u$  "and"  $\square$  "is" FIND-SET( $v$ ).ancestor

أ. أثبت أن السطر 10 يُنفَّذ مرةً واحدةً لكل زوج  $\{u, v\} \in P$ .

ب. بيّن أنه عند استدعاء  $LCA(u)$  يكون عدد المجموعات في بنية معطيات المجموعات المنفصلة مساوياً لعمق  $u$  في  $T$ .

ت. أثبت أن  $LCA$  يطبع السلف المشترك الأبعد لـ  $u$  و  $v$  لكل زوج  $\{u, v\} \in P$ .

ث. حلّل زمن تنفيذ  $LCA$  بافتراض أننا نستخدم تنحيز بنية معطيات المجموعات المنفصلة من المقطع 3.21.

## ملاحظات الفصل

يعود الفضل في العديد من النتائج الهامة المتعلقة ببنية معطيات المجموعات المنفصلة (جزئياً على الأقل) إلى تاريان R. E. Tarjan؛ فقد أعطى Tarjan [328, 330]، باستخدام التحليل الشجري aggregate analysis، الحدّ الأعلى المُضْمَك الأول بدلالة الدالة المعاكسة لدالة Ackermann ذات النمو البطيء جداً  $\mathcal{A}(m, n)$ . (تشبه الدالة  $\mathcal{A}_k(j)$  المُعطاة في المقطع 4.21 دالة Ackermann، وتشبه الدالة  $\alpha(n)$  الدالة المعاكسة. وتكون قيمة كلٍّ من  $\alpha(n)$  و  $\mathcal{A}(m, n)$  على الأكثر لجميع القيم المعقولة لـ  $m$  و  $n$ ). برهن Hopcroft و Ullman [5, 179] على حدٍّ أعلى  $O(m \lg n)$ . جرت ملاءمة لمعالجة في المقطع 4.21 بتحليل لاحق في Tarjan [332] الذي يعتمد بدوره على Kozen [220]. يعطي Harfst و Reingold [161] إصداراً للحد السابق لـ Tarjan يعتمد على الكمون.

يناقش Tarjan و van Leeuwen [333] متغيرات لكسبية ضغط المسار، تتضمن "طرائق المرور الواحد" التي تقدّم أحياناً عوامل ثابتة، أفضل في أدائها من الطرائق ذات المرورين. وكما في التحاليل السابقة لـ Tarjan لكسبية ضغط المسار الأساسية، فإنَّ تحاليل Tarjan و van Leeuwen هي تجميعية. بيّن Harfst و Reingold [161] فيما بعد كيف يمكن بتغيير بسيط لدالة الكمون ملاءمة تحليل ضغط المسار للمتغيرات ذات المرور الواحد. بيّن Gabow و Tarjan [121] أنّه في بعض التطبيقات، يمكن إجراء العمليات على المجموعات المنفصلة بزمن  $O(m)$ .

بيّن Tarjan [329] ضرورة وجود حدٍّ أدنى للزمن  $\Omega(m \mathcal{A}(m, n))$  تتطلبها العمليات على بنية معطيات المجموعات المنفصلة التي تحقق بعض الشروط التقنية. قام Fredman و Saks [113] فيما بعد بتعميم هذا الحد، وأظهرا أنّه في أسوأ الحالات، يجب أن يجري النفاذ إلى كلمات في الذاكرة طوطها  $\Omega(m \mathcal{A}(m, n))(\lg n)$  بشأ.





## مطبوعات الجمعية العلمية السورية للمعلوماتية

### معجم مصطلحات المعلوماتية

#### Dictionary of Information Technology Terms



- يضم 7000 مصطلح في شتى علوم للمعلوماتية.
- يحوي للمصطلح الأجنبي والمقابل بالعربي مع شرح للمصطلح باللغة العربية
- شارك في وضعه 30 باحثاً من هيئات علمية رفيعة.

### أسس لغات البرمجة

#### Essentials of Programming Languages



- أدوات للبرمجة الرمزية - الامتقراء والعودية والمُدَى
- التحديد النحوي وتجزيد المعطيات - قواعد الاختزال والبرمجة الأمرية
- المفسرات - تحرير المعاملات
- اللغات الفرضية التوجه - طراز تحرير الاستمرارات
- مفسرات تحرير الاستمرارات - الشكل الأمري وبناء المكسدس
- المساحات والمحللات النحوية - اشتقاق المترجمات

## هندسة البرمجيات - المجلد الأول والمجلد الثاني

### Software Engineering



- مقدمة - المنتج والإجرائية
- إدارة المشاريع البرمجية (مفاهيمها - مقاييس الإجرائية البرمجية والمشروع - تخطيط المشاريع البرمجية - إدارة المخاطرة - الجدولة الزمنية للمشروع - ضمان جودة البرمجيات - إدارة تشكيلة البرمجيات)
- طرائق تقليدية في هندسة البرمجيات (هندسة النظام - التحليل - التصميم - تصميم نظم الزمن الحقيقي - تقنيات اختبار البرمجيات - المقاييس التقنية للبرمجيات)
- هندسة البرمجيات الغرضية التوجه (مفاهيم ومبادئ - التحليل - التصميم - الاختبار - للمقاييس التقنية)
- مواضيع متقدمة في هندسة البرمجيات (الطرائق الصورية - الغرفة التنظيف - إعادة استخدام البرمجيات - إعادة الهندسة - هندسة برمجيات الزمن/المحدد - هندسة البرمجيات بمعمونة الحاسوب)

## الذكاء الصناعي

### Artificial Intelligence



- تدخل (ماهية الذكاء الصناعي - مناهج الذكاء الصناعي)
- الآلات التفاعلية - الشبكات العصبونية - ارتقاء الآلة - آلات الحالة - الرؤية الروبوتية
- الوكلاء التي تخطط - البحث الأعلى - انبثاق التجريبي - التخطيط والفعل - البحث البديل - البحث المعاكس
- البحث في فضاءات الحالة (حساب الفرضيات - حساب الاستدادات - نظم قواعد المعرفة - تمثيل المعارف البديهية - المحاكاة باستخدام المعارف غير المؤكدة - التعلم والفعل باستعمال شبكات بايز)
- طرائق التخطيط المعتمدة على النطق (حساب لتوقع - التخطيط)
- الاتصالات والتكامل (تعدد الوكلاء - الاتصال بين الوكلاء - بنيات الوكلاء)

## مفاهيم نظام التشغيل - الجزء الأول والجزء الثاني

### Operating System Concepts



- شحة عامة إلى نظام التشغيل (مقدمة - بنية نظام الحاسوب - بنية نظام التشغيل)
- الإجراءيات - التباين - جدولة وحدة المعالج - تزامن الإجراءيات - التوقف التام
- إدارة المخزن (إدارة الذاكرة - الذاكرة الافتراضية - نظام الملفات)
- نظم الإدخال والإخراج - بنية المخزن الواسع
- النظم الموزعة - التنسيق للموزع
- الحماية والأمن
- دراسة حالات (نظام لينكس - نظام ويندوز 2000 - نظام ويندوز XP)
- نظام FreeBSD - نظام Mach - نظام Nachos

### التعمية التطبيقية

#### Applied Cryptography

- أسس التعمية - لينتات بروتوكولات التعمية - البروتوكولات الأساسية - البروتوكولات المتوسطة - البروتوكولات المتقدمة - البروتوكولات الطلسمية
- تقنيات التعمية (طول المفتاح - إدارة المفاتيح - أنواع الخوارزميات وأنماطها - استخدام الخوارزميات)
- خوارزميات التعمية (مراجعة رياضية - مقياس تعمية للمعطيات DES - خوارزميات لبنية - معميات لبنية إضافية - ضم المعميات اللبينة - مولدات السلاسل شبه العشوائية والمعميات التسلسلية سمولدرات السلاسل العشوائية الحقيقية - قواعد البصمة الوحيدة الاتجاه - خوارزميات المفتاح العلني - خوارزميات التوقيع الرقمي بالمفتاح العلني - خوارزميات تبادل المفاتيح)



## المدخل إلى Mathematica 5.0

### Introduction To Mathematica 5.0



- لمحة تاريخية إلى Mathematica - ما هو Mathematica - المدى الواسع لاستخدامه
- بيئة العمل في Mathematica
- أساسيات Mathematica
- البرمجة بلغة Mathematica 5.0
- تطبيقات Mathematica 5.0 في التحليل العددي
  - (طريقة تصفيف المجال - طريقة القاطع - طريقة نيوتن -
  - رافسن - طريقة النقطة الثابتة - طريقة هالي - طريقة
  - مولر)

## اتصالات المعطيات والحواسيب - المجلد الأول والمجلد الثاني

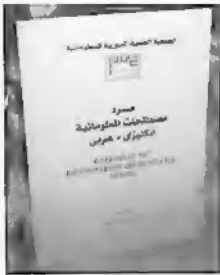
### Data And Computer Communications

- تمهيد لمواضيع الكتاب (مواضيع حفل اتصالات المعطيات واتصالات الحواسيب - مفاهيم البروتوكولات وبنائها
- تبادل المعطيات من نقطة إلى نقطة - تقانات النقل
  - التمائلي والرقمي واللاسلكي - التحكم في الوصلات -
  - التضميم.
- تبادل للمعطيات وتقانات الاتصالات للشبكات الواسعة
  - المدى (تقانات ابتدال الدارات والرمز و ATM
  - والشبكات اللاسلكية الواسعة).
- التقانات والبيانات للتشبيك على المسافات القصيرة
  - عناصر تصميم الشبكات المحلية (أوساط الإرسال، والطولوجيا، وبروتوكولات التحكم في النفاذ إلى الوسط) -
  - دراسة لبعض الشبكات المحلية للمُتَّسعة.
- الآليات والمبادئ البنيانية اللازمة لتبادل للمعطيات بين تجهيزات لمعالجة للمعطيات (حواسيب، محطات عمل، خدمات) للربط بالشبكات المحلية أو الشبكات الواسعة أو الشبكات البينية المولفة للإنترنت.





## Glossary of Information Technology Terms



- يضم 5000 مصطلح جديد
- يكوى المصطلح الأجنبي وللقابل بالعربي
- يعد مكملاً لمعجم مصطلحات المعلوماتية

مجلة الثقافة المعلوماتية



- مجلة تخصصية تعنى بالبحوث الحديثة المنشورة في أرقى الدوريات العالمية في المعلوماتية، وتتم بالترجمة الدقيقة لهذه البحوث مع المراجعة العلمية والضغط اللغوي. يصدر أربعة أعداد منها كل سنة.
- صدر منها حتى الآن 40 عدداً

تتطلب جميع مطبوعات الجمعية من المقر الرئيسي للجمعية (مجلس إدارة الجمعية) ومن جميع فروع الجمعية في المحافظات (اللجان الإدارية)

الاستعلام

هاتف : 3736156 - 2137204 - 2137205

جوال : 0933545981 - 0932503964

## المراسلة

دمشق - الجمارك - بجانب وزارة التعليم العالي

ص.ب. 33492

فاکس : 3737558 - 2137202

بريد إلكتروني: [nzhafez@scs-net.org](mailto:nzhafez@scs-net.org)

